

# NE Z8000

COMPUTADOR PESSOAL CIENTÍFICO

# Índice

## CAPÍTULO 1

### Colocando o NE-Z8000 em operação

Como utilizar este manual, quer você conheça, ou não, o BASIC

## CAPÍTULO 2

### Dizendo ao computador o que deve fazer

0, 0, DELETE, ENTER

Como introduzir dados no NE-Z8000

## CAPÍTULO 3

### Comentários sobre a linguagem BASIC

## CAPÍTULO 4

### O computador como calculadora

Instrução abordada: PRINT, como vírgula e ponto e vírgula

Funções abordadas: +, -, \*, /, \*\*

Expressões e notação científica

## CAPÍTULO 5

### Funções

Instrução abordada: RAND

Funções abordadas: ABS, SGN, SIN, COS, TAN, ASN, ACS, ATN, LN, EXP, SQR, INT, PI, RND FUNCTION

## CAPÍTULO 6

### Variáveis

Instruções abordadas: LET, CLEAR

Variáveis numéricas e simples

## CAPÍTULO 7

### Strings

Operação abordada: + (para strings)

Funções abordadas: LEN, VAL, STR\$

Strings, variáveis simples de string

## CAPÍTULO 8

### Programação

Instruções abordadas: RUN, LIST

Programas

Edição de programas empregando ◁, ▷ e EDIT

## CAPÍTULO 9

### Mais programação

Instruções abordadas: GOTO, CONT, INPUT, NEW, REM, PRINT

STOP sobre dados de entradas

BREAK

## CAPÍTULO 10

### IF

Instruções abordadas: IF, STOP

Operações abordadas: =, <, >, <=, >=, <>, AND, OR

Função abordada: NOT

## CAPÍTULO 11

### O conjunto de caracteres

Funções abordadas: CODE, CHR\$

Gráficos

## CAPÍTULO 12

### Laços (loops)

Instruções abordadas: FOR, NEXT TO, STEP

## CAPÍTULO 13

### Sub-rotinas

Instruções abordadas: GOSUB, RETURN

## CAPÍTULO 14

### Operando os programas

Fluxogramas e debug

## CAPÍTULO 15

### Armazenagem em fita

Instruções abordadas: SAVE, LOAD

## CAPÍTULO 16

### Imprimindo mais esteticamente

Instruções abordadas: CLS, SCROLL

Itens da instrução PRINT: AT, TAB

## CAPÍTULO 17

### Gráficos

Instruções abordadas: PLOT, UNPLOT

**CAPÍTULO 18**

**Tempo e movimento**

Instrução abordada: **PAUSE**

Função abordada: **INKEYS**

**CAPÍTULO 19**

**A impressora**

Instruções abordadas: **LPRINT, LLIST, COPY**

**CAPÍTULO 20**

**Sub-strings**

Operação de *slicing*, usando **TO**

**CAPÍTULO 21**

**Arrays**

Instrução abordada: **DIM**

**CAPÍTULO 22**

**Quando o computador fica repleto**

**CAPÍTULO 23**

**Contando**

Contagem binária e hexadecimal

**CAPÍTULO 24**

**Como funciona o computador**

Função de cada integrado

Instrução abordada: **POKE**

Função abordada: **PEEK**

**CAPÍTULO 25**

**Empregando linguagem de máquina**

Instrução abordada: **NEW**

Função abordada: **USR**

**CAPÍTULO 26**

**Organização da armazenagem**

**CAPÍTULO 27**

**Variáveis do sistema**

**APÊNDICES**

**A— O conjunto de caracteres**

**B— Códigos**

**C— O NE-Z8000 para os que já entendem BASIC**

# CAPÍTULO I

## Colocando o NE-Z8000 em operação

Ao desembalar o NE-Z8000, você encontrará:

1. Este Manual.

2. O Computador.

Ele tem 3 plugs de entrada (marcados 9 V DC, EAR e MIC), 1 soquete para a antena e uma parte da placa de circuito exposta, onde você pode conectar um equipamento extra. Ele não tem chave interruptora; para ligá-lo, você simplesmente o conecta no alimentador.

3. Uma Fonte de Alimentação.

Ela converte a linha de corrente alternada em uma forma usável pelo NE-Z8000. É bastante seguro, e você pode trocar os plugs do computador sem tomar choque. Você também pode conectar os plugs nos soquetes errados sem causar danos. Se quiser, pode usar o seu próprio alimentador; ele deve fornecer 9 V DC, com 700 mA, não estabilizados, e terminar em um plug de 3,5 mm com terminal positivo.

4. O Cabo Conector de Vídeo.

O qual tem aproximadamente 120 cm de comprimento e deve conectar o computador à televisão.

5. Um par de Cabos

Cada qual com tamanho aproximado de 30 cm, com um plug de 3,5 mm em cada extremidade.

Você também necessitará de uma televisão. O NE-Z8000 pode trabalhar sem ela, mas você não poderá ver o que está fazendo.

Deve ser uma televisão VHF, usando 525 linhas e 60 quadros por segundo.

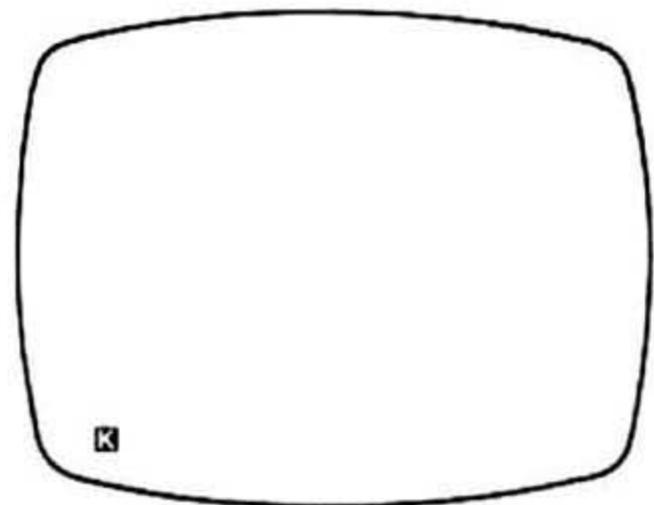
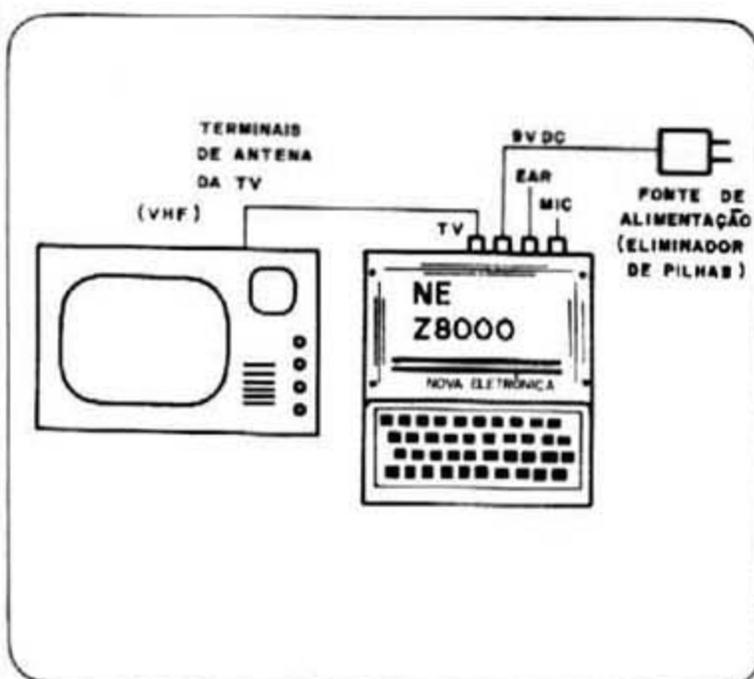
Mais tarde, você necessitará de um gravador, porque, quando o NE-Z8000 é desligado, todas as informações nele armazenadas são perdidas permanentemente.

A única maneira de mantê-las para uso posterior é gravá-las numa fita cassete. Você aprenderá a fazer isso no capítulo 15. Também poderá comprar fitas que outras pessoas preparam, e executar seus programas.

Quando você tiver tudo pronto (exceto gravador), conecte os componentes como mostra a figura.

Se a sua televisão tem duas conexões para antenas, marcadas UHF e VHF, use a que estiver marcada VHF.

Ligue a força e ligue a televisão. O NE-Z8000 opera nos canais 2 e 3 VHF. Quando o computador é conectado pela 1ª vez, mostra uma imagem como esta:



com o som de um pequeno zumbido (quando estiver usando o computador, você provavelmente vai querer baixar todo o volume da televisão).

Se sua televisão tem um controle de sintonia variável, você simplesmente tem que ajustá-lo até conseguir esta imagem.

Hoje, muitas televisões têm um botão individual para cada estação; escolha canal 2 ou 3 e sintonize.

Se você estiver impossibilitado de visualizar o cursor, lembre-se de que você pode sempre obter esta imagem tirando o plug "9 V DC IN" e colocando-o novamente. Isso deve ser feito somente como um último recurso, porque, do contrário, você perderá todas as informações que estiverem no computador.

Agora que você está familiarizado com o computador,

certamente desejará usá-lo. Se já conhece a linguagem BASIC de computador, leia o apêndice C, e use o resto do manual apenas para esclarecer algum ponto obscuro.

Se você é um principiante, a maior parte desse manual foi escrita para você. Não ignore os exercícios; muitos deles levantam pontos interessantes, os quais não são tratados no texto.

Não importa o que você faça, continue usando o computador. Se você tem a pergunta do tipo "O que ele faz, se eu disser para que ele faça isso, aquilo e aquilo?" A resposta é fácil: digite os comandos e veja. Sempre que o manual disser para digitar algo, pergunte a si mesmo "O que eu posso digitar ao invés disso?" Tente para obter sua resposta. Quanto mais você escrever do seu próprio material, melhor compreenderá o NE-Z8000.

Este equipamento gera e utiliza radiofrequência e, se não for instalado e usado adequadamente, isto é, de acordo com as instruções do fabricante, poderá provocar interferências em aparelhos de rádio e TV. O computador NE-Z8000, porém, foi exaustivamente testado, de forma a garantir uma proteção razoável contra tais interferências em instalações elétricas residenciais.

Caso o seu aparelho chegue a ocasionar algum tipo de interferência a receptores de rádio ou TV (o que pode ser comprovado ao se ligar e desligar o computador, em seguida), recomenda-se a adoção de uma das seguintes alternativas:

- Reorientação da antena receptora
- Mudança da posição do computador em relação ao aparelho afetado
- Afastar o computador o mais possível do aparelho afetado
- Ligar o computador a uma outra tomada da casa, de forma a conectá-lo a um outro circuito elétrico, que não o do aparelho afetado

Caso seja necessário, o usuário poderá consultar um técnico experiente em rádio e TV, a fim de obter mais algumas informações úteis, de acordo com seu caso particular.

## CAPÍTULO II

### Dizendo ao computador o que fazer

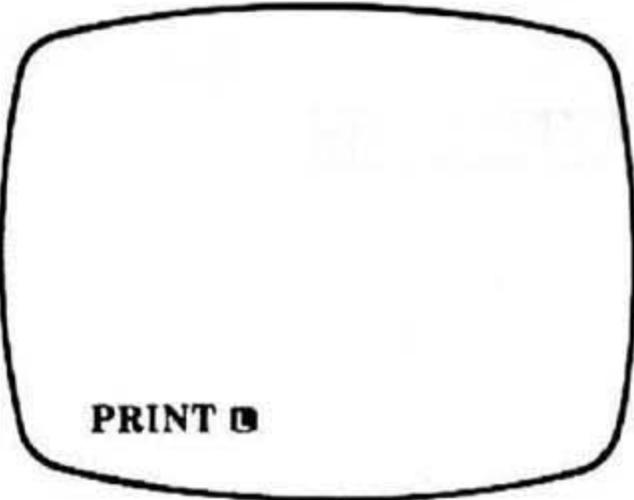
Agora que você tem na tela a imagem apresentada no capítulo 1, você pode digitar instruções especiais de computadores. Por exemplo:

**PRINT 2 + 2**

faz o computador processar a soma  $2 + 2$  e exibir (PRINT) o resultado na tela. Uma instrução como esta, na qual o computador executa de imediato, é tecnicamente chamada **comando BASIC**. (Há outras instruções BASIC usadas nos programas — elas não são executadas de imediato. Serão vistas nos capítulos de 8 a 13.)

Para digitar um comando:

1. Primeiro digite **PRINT**. Mas, como você pode ver, apesar do teclado ter uma tecla para cada letra, você não necessita digitar a palavra P, R, I, N, T. Tão logo você pressione P, toda a palavra aparecerá na tela, seguida de um espaço, e a tela ficará dessa maneira:



**PRINT** **⏏**

A razão disso é que no começo de cada comando o computador está esperando uma *palavra-chave* — uma palavra que especifique qual o tipo de comando. As palavras-chaves são escritas em cima das teclas, e você verá que **"PRINT"** aparecerá sobre a tecla P, de tal maneira que para conseguir o comando **"PRINT"** basta pressionar P.

O computador informa que está esperando uma palavra-chave através do **⏏** que aparece no vídeo. Quase sempre existe uma letra em preto e branco (*video inverso*) que pode ser um **⏏** ou um **⏏** (ou poderemos ver ainda as letras, **⏏** ou **⏏**), chamado **cursor**. O **⏏** significa que qualquer tecla a ser pressionada deve ser tomada como a palavra-chave. Como você viu, após ter pressionado o P para **PRINT**, o **⏏** mudou para um **⏏**.

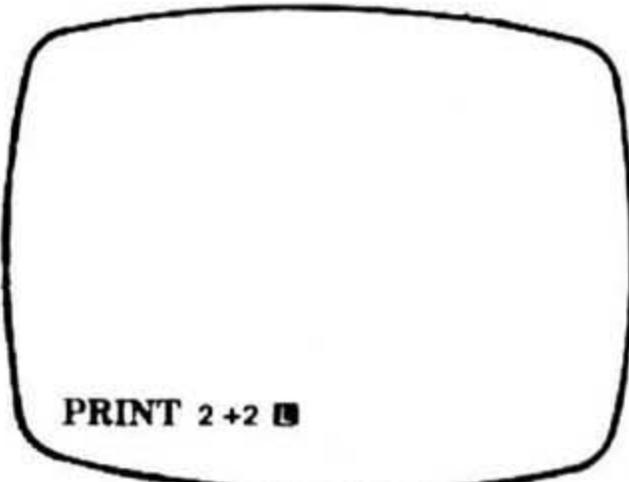
Esse sistema de pressionar apenas uma tecla para obter mais de um símbolo, é muito usado no NE-Z8000. No resto desse manual, palavras que tenham suas próprias teclas são impressas em negrito. Palavras-chaves não podem ser digitadas letra por letra.

2. Agora digite 2. Isso não deve causar nenhum problema. Novamente você deverá ver o 2 aparecer na tela, e a letra L avançar um espaço.

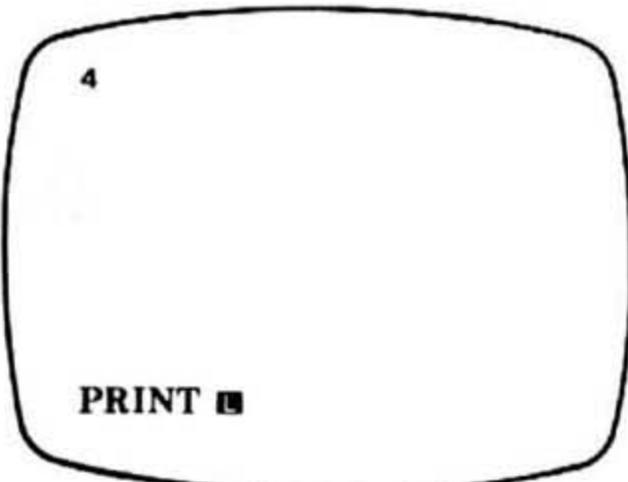
Note também que um espaço é automaticamente colocado entre o **PRINT** e o 2, para ficar mais nítido. Isso é feito o máximo possível, de tal maneira que você dificilmente tenha que digitar um espaço. Se você digitar um espaço, ele aparecerá na tela, mas não afetará o comando.

3. Agora digite +. Esse é um caractere **maiúsculo** (tais caracteres estão marcados em vermelho — a cor da tecla **SHIFT** no canto direito superior de cada tecla), e para conseguir "+" você deve pressionar a tecla **SHIFT**, e, ao mesmo tempo, pressionar a tecla que contém os símbolos **LIST**, **K**, + e **LEN**.

4. Agora digite 2 novamente. A tela ficará dessa maneira:



PRINT 2 + 2 █



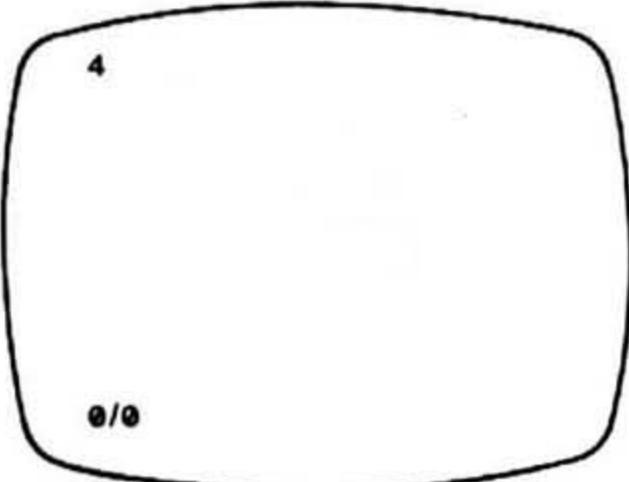
4 █

5. Agora pressione **ENTER**, a tecla **FUNCTION ENTER**. Você pressiona esta tecla quando finaliza a linha de instrução. O computador então processará. No nosso caso, a tela muda para

O cursor pode também ser usado para corrigir erros: digite ++2, para obter

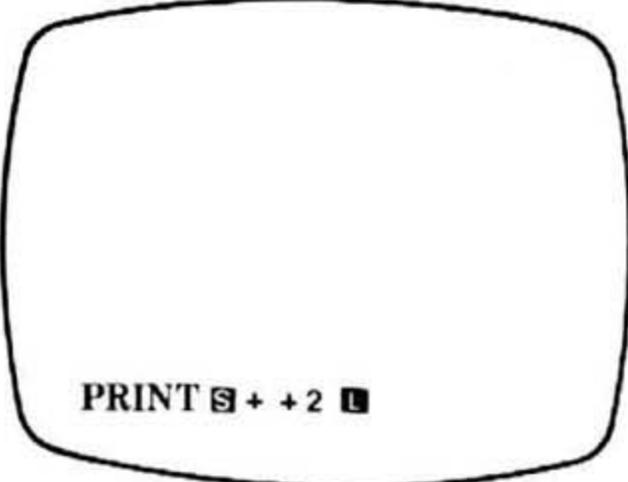
PRINT + + 2

na linha inferior, quando você pressionar **ENTER**, você obtém:



4 █

0/0



PRINT █ + + 2 █

4 é a resposta — mas é claro que você não precisa comprar um computador para descobrir isso.

0/0 é a **denotação** através da qual ele informa como o resultado foi conseguido (note que o zero está escrito cortado por uma barra para que seja diferenciado do O maiúsculo. Isto é muito comum na área de computação).

O primeiro 0 significa "OK, sem problemas". (No apêndice B há uma lista de outros códigos de **denotações** que podem aparecer; por exemplo, se alguma coisa der errado.) O segundo 0 significa que "a última coisa a ser feita foi a linha 0". Você verá mais tarde, quando começar a escrever programas, que para uma instrução pode ser dado um número, para podermos guardá-la para execução posterior: ela passa então a ser uma **linha de programa**. Comandos, na realidade, não têm números, mas para conveniência das denotações o computador trata-os como linha 0.

Você deve considerar a denotação como se ela estivesse escondendo o cursor █ — se você pressionar P para **PRINT**, a denotação desaparecerá e a tela mudará para

O █ é o **indicador de erro de sintaxe** (sintaxe é a gramática de mensagens, dizendo o que é e o que não é permitido); ele mostra que o computador entendeu a linha até "PRINT", mas após isso há algo errado.

O que você deseja fazer, logicamente, é cancelar o primeiro + trocando-o por um — digamos — 3. Primeiramente deve-se mover o cursor de tal maneira que ele fique à direita do primeiro +; há duas teclas, 0 e 0 (SHIFT 5 e SHIFT 8), as quais movem o cursor para a direita e para a esquerda. Pressionando **SHIFT**, pressione a tecla 0 duas vezes. Isto move o cursor 2 espaços para a esquerda, e ficamos assim:

PRINT + █ + 2

Agora pressione a tecla **DELETE** (SHIFT Ø), e você obterá:

**PRINT** [ ] + 2

**DELETE** remove o caractere ou palavra-chave imediatamente à esquerda do cursor.

Se você agora pressionar 3 em "3", será inserido, novamente à esquerda do cursor, dando

**PRINT** 3 [ ] + 2

e pressionando **ENTER** dá a resposta (5).

A tecla Ø (SHIFT 8) funciona exatamente igual a tecla φ, sendo que move o cursor para a direita ao invés de para a esquerda.

### Resumo

Este capítulo refere-se à parte de como digitar mensagens para o NE-Z8000, explicando o sistema de digitação simples.

Os cursores **K** e **L**,

Denotações,

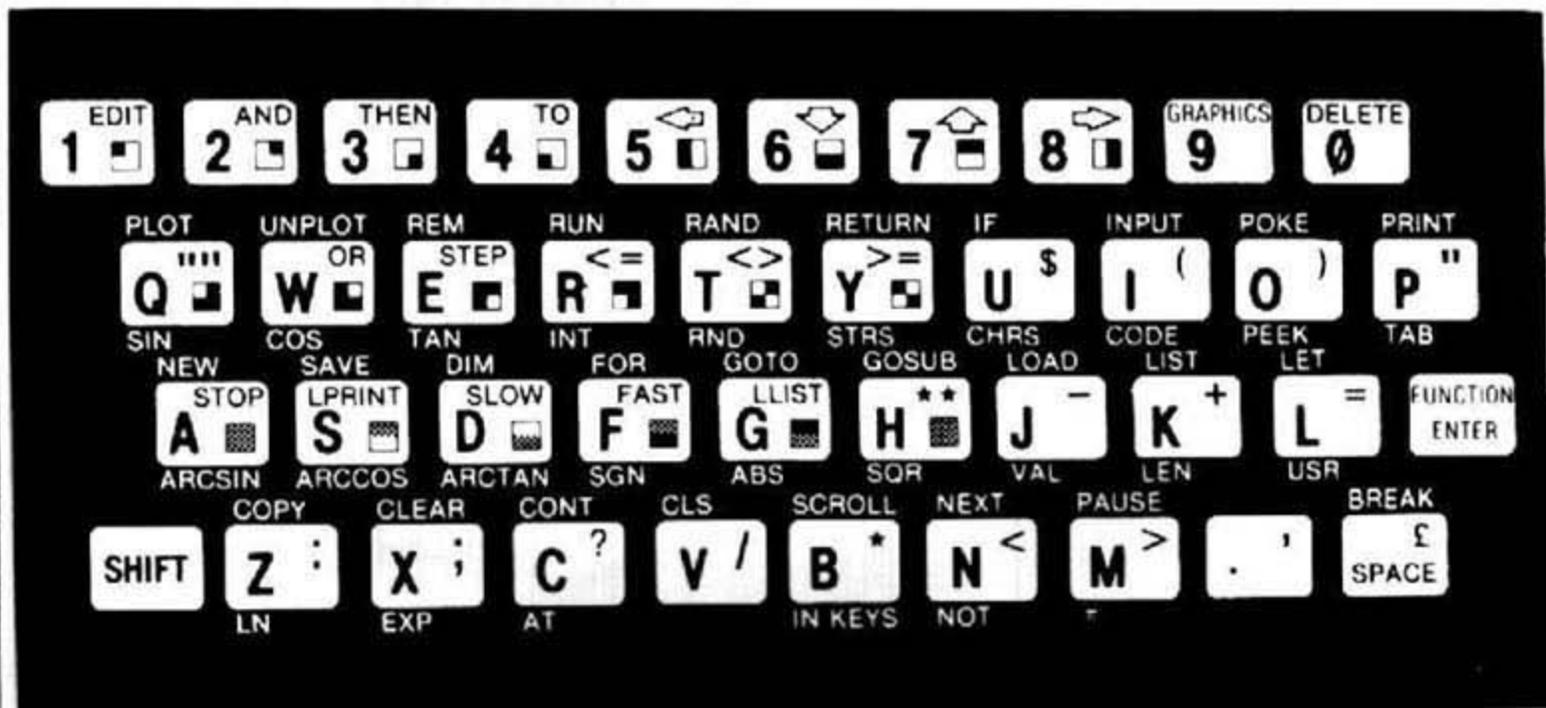
Indicador de erros de sintaxe, **S**

Como corrigir erros usando Ø, φ e **DELETE**.

### O Teclado

Se você estiver familiarizado com máquinas de escrever, você pode usar esta ilustração para aprender onde as letras se encontram. Lembre-se de que para caracteres maiúsculos você deve pressionar a tecla **SHIFT** ao mesmo tempo que pressiona outra tecla. Não confunda o dígito Ø com a letra O

Aqui está uma reprodução do teclado.



## CAPÍTULO III

### Comentários sobre a linguagem Basic

As instruções que você digita no computador estão em uma linguagem de computador chamada **BASIC** (Beginners' All-purpose Symbolic Instruction Code). O computador, na realidade, faz muitos esforços para transformar as instruções BASIC em suas próprias operações rudimentares. A linguagem BASIC contém bastante palavras em inglês (como PRINT) para torná-las fáceis de memorizar.

O BASIC foi desenvolvido no Dartmouth College em New Hampshire, no ano de 1964, e desde então ela tornou-se a linguagem de computador mais usada por principiantes e hobistas. Isto se deve em grande parte ao BASIC ser bem adaptado ao uso **on-line**, onde o usuário digita alguma coisa e o computador responde imediatamente.

Há outras linguagens de programação — tais como o PASCAL — com estrutura mais regular e muito mais poderosa que o BASIC, mas apenas algumas, como APL, são fáceis de usar on-line. Algumas outras que devem ser mencionadas são FORTRAN, COBOL, RPG e PLI.

Muitas revistas de computadores pessoais publicam programas em BASIC, e vale a pena dar uma olhada para procurar algumas idéias. Você certamente terá que alterá-las um pouco, pois todo computador que usa a linguagem BASIC tem sua própria versão, diferente de todas as outras.

# CAPÍTULO IV

## O computador como calculadora

Ligue o computador. Você agora pode usá-lo como uma calculadora de bolso, como foi descrito no capítulo 2: digite **PRINT**, e então, em seguida, qualquer coisa que deseje resolver, e por último **ENTER**. (Nós não nos cansaremos de dizer para digitar **ENTER**.)

O NE-Z8000 não apenas adiciona, mas também subtrai, multiplica (usando o asterisco em vez do sinal usual de vezes — isto é muito comum em computadores) e divide (usando / em vez de +). Experimente.

+, —, \* e / são **operações** e os números por eles operados são chamados **operandos**.

O computador pode também elevar um número à potência de outro, usando o sinal \*\* (SHIFT H). Digite:

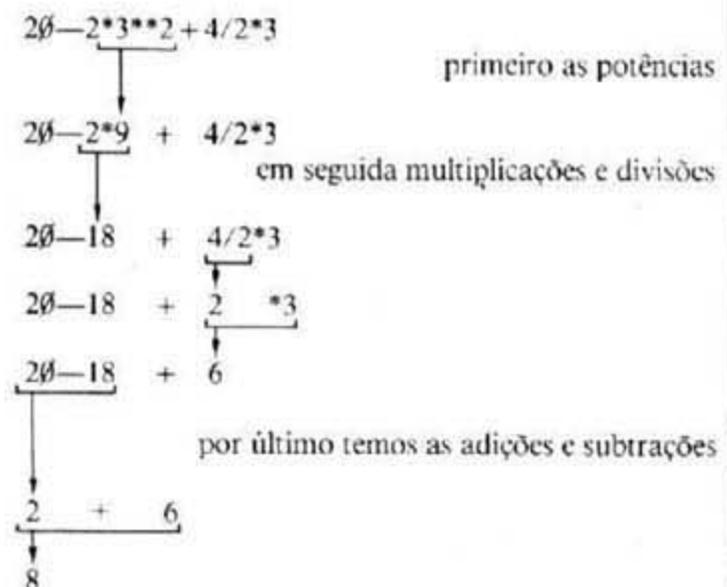
**PRINT 2\*\*3**

e você obterá a resposta 8 ( $2^3$ , ou 2 ao cubo).

O NE-Z8000 também processará combinações das operações. Por exemplo:

**PRINT 20-2\*3\*\*2+4/2\*3**

dá a resposta 8. Ele faz isso de maneira indireta, porque primeiro ele faz a exponenciação (\*\*) na ordem, da esquerda para a direita; depois todas as multiplicações e divisões (\* e /), novamente da esquerda para a direita, e, então, executa as adições e subtrações (+ e —), ainda da esquerda para a direita. Assim, o nosso exemplo é executado nos seguintes estágios:



Nós formalizamos isto dando para cada operação uma **prioridade**, um número entre 1 e 16. As operações com as mais altas prioridades são executadas primeiro, e as de igual prioridade são executadas na ordem que aparece, da esquerda para a direita.

- \*\* tem prioridade 10
- \* e / tem prioridade 8
- + e — tem prioridade 6

Quando — é usado para deixar algo negativo, como por exemplo quando você escreve —1, ele tem prioridade 9. Este é o menos **unário**, o qual difere do menos **binário** em 3—1. (Uma operação unária tem apenas um operando, enquanto uma operação binária tem dois.) Note que no NE-Z8000 não pode ser usado + como uma operação unária.

Esta ordem é absolutamente rígida, mas você pode alterá-la através do uso de parênteses: qualquer coisa entre parênteses é executada primeiro e depois passa a ser considerada como um número simples de tal maneira que

**PRINT 3\*2+2**

dá como resposta  $6+2=8$ , mas

**PRINT 3\*(2+2)**

dá como resposta  $3*4=12$ .

Uma combinação como esta é chamada **expressão** — neste caso, uma expressão **aritmética** ou **numérica**, porque o resultado é um número. Geralmente, quando o computador está esperando a entrada de um número, você pode digitar uma expressão e ele encontrará o resultado.

Você pode escrever números com ponto decimal (o ponto equivale à vírgula) e também pode usar notação científica, como é muito comum em calculadoras de bolso. Desta maneira, após um número comum (com ou sem ponto decimal — vírgula), você pode escrever o **expoente**, o qual consiste da letra **E**, e o sinal **+** ou **-**, e um número sem ponto decimal. O **E** aqui significa **"\* 10\*\*"**, ("vezes 10 elevado à potência de"). Assim, temos:

$$2.34E0 = 2.34 * 10^{**0} = 2.34$$

$$2.34E3 = 2.34 * 10^{**3} = 2340$$

$$2.34E-2 = 2.34 * 10^{**-2} = 0.0234 \text{ e assim por diante.}$$

(Tente fazer isso no NE-Z8000.)

Você compreenderá isso melhor se imaginar o expoente como uma mudança do ponto decimal para a direita (se o expoente for positivo) ou para a esquerda (se o expoente for negativo).

Você também pode fazer um **PRINT** de mais de um número ao mesmo tempo, separando-os com vírgulas (,) ou ponto e vírgula (; é SHIFT X). Se você usar vírgula, o próximo número será impresso ou na margem esquerda ou no meio da linha da coluna 16. Se você usar um ponto e vírgula, o próximo número será impresso imediatamente após o último. Tente

**PRINT 1;2;3;4;5;6;7;8;9;10**

**PRINT 1,2,3,4,5,6,7,8,9,10**

para ver as diferenças. Você pode misturar vírgulas e ponto e vírgulas na mesma instrução **PRINT**.

#### Resumo

Instruções: **PRINT**, com vírgulas ou ponto e vírgulas.

Funções: **+**, **-**, **\***, **/**, **\*\***

Expressões, notação científica

#### Exercícios:

1. Tente

**PRINT 2.34E0**

**PRINT 2.34E1**

**PRINT 2.34E2**

até

**PRINT 2.34E15**

Você notará também que o NE-Z8000 também utiliza a notação científica. Isso porque ele nunca usa mais de 14 espaços para escrever um número. Da mesma forma, tente

**PRINT 2.34E-1**

**PRINT 2.34E-2**

e assim por diante.

2. Tente

**PRINT 1,,2,,3,,4,,,5**

Uma vírgula sempre ocupará o lugar do próximo número. Agora tente

**PRINT 1::2:::3:::4:::5**

Porque uma seqüência de ponto e vírgula não é diferente de apenas um?

3. **PRINT** fornece apenas 8 dígitos significativos. Tente

**PRINT 4294967295, 4294967295 — 429E7**

Isto prova que o computador pode trabalhar com todos estes dígitos apesar de não estar preparado para imprimi-los de uma vez.

4. Se você tem uma tabela de logaritmos, teste esta regra:

Elevar 10 à potência de um número é o mesmo que tirar o antilog daquele número.

Por exemplo, digite

**PRINT 10\*\*0.3010**

e veja o antilog de  $0.3010$ . Por que o resultado não é exatamente igual?

5. O NE-Z8000 utiliza aritmética de **ponto flutuante**, o que significa que ele mantém separados os dígitos de um número (a **mantissa**) e a posição do ponto (o **expoente**). O resultado nem sempre é exato, mesmo para números inteiros.

Digite

**PRINT 1E10 + 1 — 1E10, 1E10 — 1E10 + 1**

Os números são mantidos numa precisão de  $9\frac{1}{2}$  dígitos, assim  $1E10$  é muito grande para ser mantido exatamente correto. O erro (falta de precisão) (na verdade 2) é mais que 1, assim os números  $1E10$  e  $1E10 + 1$  parecem ser iguais para o computador.

Para um exemplo mais detalhado, digite:

**PRINT 5E9 + 1 — 5E9**

Aqui, o erro em  $5E9$  é apenas  $\pm 1$ , e o 1 a ser adicionado, na realidade, torna-se **arredondado** para 2. Aqui os números  $5E9 + 1$  e  $5E9 + 2$  parecem ser iguais para o computador.

O maior inteiro que pode ser tratado com precisão absoluta é  $2^{32} - 1$  (4.294.967.295).

# CAPÍTULO V

## Funções

Matematicamente, uma função é uma regra para fornecer um número (o **resultado**) em troca de um outro (o **argumento**, ou **operando**) e assim é realmente uma operação unária. O NE-Z8000 tem algumas dessas funções já definidas internamente. Seus nomes são palavras escritas abaixo das teclas; **SQR**, por exemplo, é a conhecida função raiz quadrada, e

**PRINT SQR 9**

tem como resultado 3, a raiz quadrada de 9 (para obter **SQR** você deve primeiro pressionar a tecla **FUNCTION** — que é obtida pressionando-se **SHIFT** e **ENTER** ao mesmo tempo. Isso muda o cursor para **F**. Agora pressione a tecla **SQR** (H): **SQR** aparece na tela e o cursor volta novamente para **F**. O mesmo método funciona para todas as palavras que estão escritas abaixo das teclas, as quais quase todas são nomes de funções.

**PRINT SQR 2**

Você pode testar a precisão da resposta da seguinte maneira:

**PRINT SQR 2\*SQR 2**

a qual tem que ser 2. Note que ambos **SQRs** são executados antes da **\***, e na verdade todas as funções (exceto uma — **NOT**) são executadas antes das cinco operações **+**, **-**, **\***, **/** e **\*\***. Esta regra pode ser alterada através do uso de parênteses:

**PRINT SQR (2\*2)**

o resultado é 2.

Aqui estão mais algumas funções (há uma lista completa no apêndice C)

**SGN** A função signal (algumas vezes chamada **signum** para não confundir com **SIN**). O resultado é **-1**, **0** ou **+1**, dependendo do argumento ser negativo, zero ou positivo.

**ABS** O valor absoluto, ou módulo. O resultado é sempre o argumento positivo, de tal maneira que:

$$\text{ABS } -3.2 = \text{ABS } 3.2 = 3.2$$

**SIN**  
**COS**  
**TAN**  
**ASN**  
**ACS**  
**ATN**

aresin  
arccos  
arctan

As funções trigonométricas trabalham em radianos, não em graus.

**LN**  
**EXP**  
**SQR**  
**INT**

logaritmo natural (base 2,718281828 ...; aliás e)  
função exponencial

raiz quadrada

parte inteira. Esta função sempre arredonda para menos; assim, **INT 3.9 = 3** e **INT -3.9 = -4**. (Um **inteiro** é um número não fracionário, positivo ou negativo.)

**PI**  
**RND**

$\pi = 3,1415927$  (**PI** não tem argumento).

**RND** também não tem argumento. Ele gera um número randômico entre 0 (valor que pode ser assumido) e 1 (o qual não pode assumir).

Usando a terminologia do último capítulo, todas as funções, exceto **PI** e **RND** são unárias (funções com um único operando) com prioridade 11. (**PI** e **RND** são operações nulas, porque não têm operandos.)

As funções trigonométricas, bem como **EXP**, **LN** e **SQR**, são geralmente calculadas com precisão de 8 dígitos.

**RND** e **RAND** estão na mesma tecla, mas ao passo que **RND** é uma função, **RAND** é uma palavra chave, como **PRINT**. **RAND** é usado para controlar a randomicidade de **RND**.

**RND** não é verdadeiramente randômico, pois segue uma seqüência de 65536 números tão misturados que parece ser randômico (**RND** é um **pseudo-randômico**). Você pode usar **RAND** para inicializar **RND** em diferentes lugares na seqüência, digitando **RAND** e um número entre 1 e 65535, e então **ENTER**. Não é importante saber onde o número dado inicializa **RND**, mas sim que o mesmo número após **RAND** inicializará **RND** sempre no mesmo lugar. Por exemplo, digite

**RAND 1**

(e ENTER)

**PRINT RND**

e então

**PRINT RND**

ponha isso em "looping" várias vezes. (Lembre-se de usar **FUNCTION** para obter **RND**.) O resultado de **RND** será sempre 0.0022735596, não uma sequência randômica.

**RAND 0**

(o zero não precisa ser digitado) age ligeiramente diferente: determina onde inicializar **RND** ao longo do tempo que a televisão estiver ligada, e este é genuinamente randômico.

**Resumo**

Instrução: **RAND**

Funções: **SGN, ABS, SIN, COS, TAN, ASN, ACS, ATN, LN, EXP, SQR, INT, PI, RND**

**Exercícios**

1. Para obter os logaritmos (base 10), que você encontrará nas tabelas de log, divida o logaritmo natural por **LN 10**. Por exemplo, para encontrar  $\log 2$ ,

**PRINT LN 2/LN 10**

a resposta é 0.30103.

Tente fazer multiplicações e divisões usando o **NE-Z8000** como um conjunto de tabelas de logaritmos desta maneira (para antilogs, veja o capítulo 2 — exercício 3). Confira as respostas usando \* e /. A maneira direta é mais precisa.

2. **EXP** e **LN** são funções **inversas** no sentido de que se você aplica uma e então a outra, você consegue de volta o seu número original. Por exemplo,

**LN EXP 2 = EXP LN 2 = 2**

O mesmo acontece para **SIN** e **ASN**, para **COS** e **ACS**, e para **TAN** e **ATN**. Você pode usar isto para testar a precisão com a qual o computador trabalha estas funções.

3.  $\pi$  radianos é igual  $180^\circ$ . Para converter graus em radianos, divida por 180 e multiplique por  $\pi$ . Assim

**PRINT TAN (45/180\*PI)**

que dá  $\tan 45^\circ$  (1).

Para converter de radianos para graus, divida por  $\pi$  e multiplique por 180.

4. Tente

algumas vezes para verificar como o resultado varia. Você detectou algum padrão? (Provavelmente não.)

Como você usaria **RND** e **INT** para obter um número inteiro entre 1 e 6, se representasse um risco de vida? (Resposta: **INT (RND \*6) + 1**.)

5. Teste esta regra:

Suponha que você escolha um número entre 1 e 872 e digite

**RAND** e seu número (e **ENTER**)

O próximo valor de **RND** será

$(75 * (\text{seu número} + 1) - 1) / 65536$

6. (Para matemáticos apenas.)

$p$  é o (maior) número primo, e  $a$  é a raiz primitiva módulo  $p$ .

Então, se  $b_i$  é o residuo de  $a^i$  módulo  $p$  ( $1 \leq i < p-1$ ), a sequência

$$\frac{b_i - 1}{p - 1}$$

é uma sequência cíclica de um número distinto  $p-1$  na faixa de 0 a 1 (excluindo 1). Escolhendo convenientemente, você pode fazer isto parecer bastante randômico.

65537 é um número primo de Mersenne,  $2^{16} - 1$ . Use isso, com a lei de Gauss da reciprocidade quadrática, para mostrar que 75 é a raiz primitiva módulo 65537.

O **NE-Z8000** usa  $p = 65537$  e  $a = 75$ , e armazena  $b_i - 1$  na memória. A função **RND** envolve a reposição de  $b_i - 1$  na memória por  $b_{i+1} - 1$ , e a geração do resultado  $(b_{i+1} - 1) / (p - 1)$ . **RAND**  $n$  (com  $1 \leq n \leq 65535$ ) faz  $b_i$  igual a  $n + 1$ .

7. **INT** sempre arredonda para menos. Para arredondar ao inteiro mais próximo, adicione 0.5 primeiramente. Por exemplo,

$$\begin{aligned} \text{INT} (2.9 + 0.5) &= 3 & \text{INT} (2.4 + 0.5) &= 2 \\ \text{INT} (-2.9 + 0.5) &= -3 & \text{INT} (-2.4 + 0.5) &= -2 \end{aligned}$$

Compare estes resultados com os que você obtém quando não adiciona 0.5.

8. Tente

**PRINT PI, PI-3, PI-3,1, PI-3,14, PI-3, 141**

Isso mostra a precisão com a qual o computador armazena  $\pi$ .

## CAPÍTULO VI

### Variáveis

Minha calculadora de bolso pode armazenar um número e processá-lo mais tarde. O NE-Z8000 faz isso?

Sim, na realidade ele pode, literalmente, armazenar centenas deles, usando a instrução **LET**. Suponha que os ovos custem Cr\$ 58,00 a dúzia, e você deseja armazenar isso. Digite

```
LET OVOS = 58      (e ENTER, lógico)
```

Agora, isso causou uma série de acontecimentos. Primeiramente, o computador reservou um local para armazenamento de um número; segundo, deu a este local o nome de "OVOS", assim você pode ter acesso a ele posteriormente; esta combinação de local de armazenamento e nome é chamada **variável**. Terceiro, ele armazenou o número 58 no local: nós dizemos que ele **relacionou** o valor 58 à variável [cujo nome é] OVOS. OVOS é uma variável numérica, porque seu valor é um número.

Você quer saber quanto custam os ovos? Digite

```
PRINT OVOS
```

Se você quer saber quanto custa meia dúzia de ovos, digite

```
PRINT OVOS/2
```

Suponha que o preço dos ovos subiu para Cr\$ 61,00 a dúzia. Digite

```
LET OVOS = 61
```

Isso atualiza o valor de 58 para 61. Confira digitando

```
PRINT OVOS
```

Agora digite

```
PRINT LEITE
```

Você obterá a denotação 2/Ø. Segundo o apêndice B, código 2 significa "variável não encontrada" — não foi dado ao computador o valor da variável "LEITE". Digite

```
LET LEITE = 18,5
```

E digite

```
PRINT LEITE
```

novamente.

Você pode usar qualquer letra ou dígito no nome de uma variável, desde que a primeira seja uma letra. (Você pode também colocar espaços, entretanto eles não serão considerados parte do nome.)

Por exemplo, eis alguns conjuntos admitidos como nomes de variáveis:

```
LJQBP
```

```
X
```

```
A3
```

AREA UM (isto e AREAUM são tratados como iguais)

mas estes não são admitidos:

```
3B (começa com um dígito)
```

```
TAL? (? não é uma letra ou um dígito)
```

```
☒ (caracter inverso de vídeo não é permitido)
```

```
ALPHA-1 (- não é letra nem dígito)
```

Agora digite:

```
CLEAR
```

A função do **CLEAR** é liberar todo espaço de armazenamento que foi reservado para variáveis. Tente digitando

## PRINT OVOS

novamente. Você obterá denotação 2 (variável não encontrada).

**Nota:** em algumas versões do BASIC é permitido omitir o **LET** e simplesmente digitar

```
OVOS = 58
```

Isso **não** é permitido no NE-Z8000 em nenhum caso; você veria que é impossível digitar.

Também em algumas versões, apenas os dois primeiros caracteres de um nome são conferidos, de tal maneira que **RADIO 3** e **RADIO 33** seriam considerados o mesmo nome; e em alguns outros, um nome de variável deve ser uma letra seguida de um dígito. Nenhuma dessas restrições referem-se ao NE-Z8000.

Novamente, em algumas versões do BASIC, se uma variável não apareceu do lado esquerdo de uma instrução **LET** é considerada como valor  $\emptyset$ . Como você viu acima com **PRINT LEITE**, isso não acontece no NE-Z8000.

### Resumo

Variáveis

Instruções: **LET, CLEAR**

### Exercícios

1. Se você não está familiarizado com elevação à potência (\*\*, SHIFT H), então faça este exercício.

No seu nível mais elementar, "**A\*\*B**" significa "A multiplicado por ele mesmo B vezes"; mas isso obviamente só faz sentido se B é um número inteiro positivo. Para encontrar uma maneira que funcione para outros valores de B, nós consideramos a regra

$$A^{**}(B+C) = A^{**}B * A^{**}C$$

Você não precisa muito para se convencer de que isto funciona, quando B e C são positivos, mas se decidirmos que nós queremos que funcione mesmo quando eles não são, nos encontramos inclinados a aceitar que

$$\begin{aligned} A^{**}\emptyset &= 1 \\ A^{**}(-B) &= 1 / A^{**}B \\ A^{**}(1/B) &= B \text{ raiz de } A \\ A^{**}(B*C) &= (A^{**}B)^{**}C \end{aligned}$$

Se você nunca viu isso antes, não tente assimilar tudo; apenas lembre-se de que

$$\begin{aligned} A^{**}-1 &= 1/A \\ A^{**}(1/2) &= \text{raiz quadrada de } A \end{aligned}$$

e talvez, quando você se familiarizar com isso, o resto começará a fazer sentido.

Experimente com tudo isso dizer ao computador para imprimir várias expressões contendo \*\*, isto é,

```
PRINT 3**(2+empty),3**2*3**empty
PRINT 4** -1,1/4
```

2. Digite

```
LET E = EXP 1
```

Agora E tem o valor 2,718281828 ..., a base do logaritmo natural. Teste a regra

**EXP** de um número = **E\*\*** o número

para vários números.

# CAPÍTULO VII

## Strings

Uma coisa que o NE-Z8000 pode fazer que calculadoras de bolso não podem é lidar com textos. Digite

```
PRINT "OLÁ. EU SOU SEU NE-Z8000".  
      ("é SHIFT P)
```

O texto entre aspas é chamado de **strings** (significa uma sequência de caracteres) e pode conter qualquer caractere que você quiser, exceto aspas ("). (Mas você pode usar as chamadas aspas reversas, "" (SHIFT Q), e elas serão impressas como ".)

Um erro comum de digitação de strings é esquecer uma das aspas — isso fara com que o **S**, indicador de erro de sintaxe, entre em ação.

Se você estiver imprimindo números, pode usar strings para explicar o que os números significam. Por exemplo, digite

```
LET OVOS = 61
```

e então

```
PRINT "O PREÇO DOS OVOS É ";OVOS;"  
CRUZEIROS A DÚZIA."
```

(Não se preocupe se a digitação exceder a linha.)

Essa instrução imprime três coisas (itens de impressão), que é a string "O PREÇO DOS OVOS É", o número 61 (o valor da variável OVOS), e a string "CRUZEIROS A DÚZIA". Na realidade, você pode fazer um **PRINT** de qualquer número de itens e de qualquer mistura de strings e números (ou expressões). Note que os espaços de uma string fazem parte da mesma forma que as letras. Eles não são ignorados, nem mesmo os finais.

Há uma porção de coisas que você pode fazer com as strings:

1. Você pode associá-las a variáveis. Entretanto, o nome da variável deve ser especial para mostrar que seu valor é uma string e não um número. O nome pode ser apenas uma letra seguida de \$ (SHIFT U). Por exemplo, digite

```
LET AS = "OMELETE DE QUEIJO"
```

e

```
PRINT AS
```

2. Você pode também juntá-los. Isso é chamado de **concatenação**, significando "junção", que é exatamente o que ele faz. Tente

```
PRINT "BACON" + " E OVOS"
```

Você não pode subtrair, multiplicar ou dividir strings, ou elevá-las à potência.

3. Você pode aplicar algumas funções em strings para obter números, e vice e versa.

**LEN** é aplicado a uma string e o resultado é seu comprimento. Por exemplo **LEN "QUEIJO" = 6**.

**VAL** é aplicado a uma string formada por números ou expressões numéricas. O que o **VAL** faz é permitir que estes valores sejam tratados como números e não como *string*. Por exemplo (se  $A = 9$ ): **VAL "1/2 + SQRA" = 3,5**. Se a string na qual **VAL** é aplicada contém variáveis, então duas regras devem ser obedecidas.

- (I) Se a função **VAL** for parte de uma expressão maior, ela deve ser o primeiro item, isto é, **1Ø LET X = 7 + VAL "Y"** deve ser mudado para **1Ø LET X = VAL "Y" + 7**.
- (II) **VAL** só pode aparecer na primeira coordenada de uma instrução **PRINT AT**, **PLOT** ou **UNPLOT** (veja capítulos 17 e 18), isto é  
**1Ø PLOT 5, VAL "X"** deve ser mudado para  
**1Ø LET Y = VAL "X"**  
**15 PLOT 5, Y**



## CAPÍTULO VIII

### Programação

E agora você finalmente pode escrever um programa de computador. Desligue e ligue o computador para certificar-se de que ele está limpo. Agora digite

```
10 LET MANTEIGA = 75      (e ENTER)
```

e a tela ficará assim:



É diferente do que aconteceu com OVOS no capítulo 6; se você digitar

```
PRINT MANTEIGA
```

você verá (denotação 2) que a variável MANTEIGA não foi implementada. (Pressione ENTER novamente e a tela deve voltar a ficar como anteriormente na figura.)

Porque a instrução LET tinha um número 10 antes dela, o computador não a executou imediatamente, mas salvou-a para uso posterior. 10 é o número da linha usado para

referenciá-la da mesma maneira que os nomes são usados para variáveis. Um conjunto dessas instruções armazenadas é chamado de programa. Agora digite

```
20 PRINT MANTEIGA
```

A tela deve ficar assim:



Esta é uma listagem de seu programa. Para fazer com que o programa seja executado, digite

```
RUN      (e ENTER, é claro)
```

e a resposta 75 aparecerá no canto superior esquerdo da tela. No canto inferior esquerdo você verá a denotação 0/20. 0, como você sabe, significa "OK, sem problemas", e 20 é o número da linha onde a execução terminou. Pressione ENTER e a listagem reaparecerá.

Note que as instruções foram executadas na ordem dos números de suas linhas.

Agora, suponha que de repente você se lembra que também necessita gravar o preço do fermento. Digite

```
15 LET FERMENTO = 40
```

Isto seria muito mais difícil se as duas primeiras linhas estivessem numeradas 1 e 2 em vez de 10 e 20 (os números das linhas devem ser inteiros e estar entre 1 e 9999). Eis porque, quando se está digitando um programa, é bom deixar espaços nos números das linhas.

Agora você necessita mudar a linha 20 para

```
20 PRINT MANTEIGA, FERMENTO
```

Você poderia digitar toda a linha, mas há uma maneira de usar o que já está lá. Vê aquele pequeno  próximo à linha 15? Este é o **cursor de programa**, e a linha para a qual ele aponta é a **linha corrente**. Pressione a tecla **←** (SHIFT 6), e ele moverá para baixo, para a linha 20. (→ move o cursor para cima novamente.) Agora pressione a tecla **EDIT** (SHIFT 1) e uma cópia da linha 20 virá para o final da tela. Pressione a tecla **↵** 9 vezes de tal maneira que o cursor se mova para o fim da linha e então digite

```
    ,FERMENTO          (sem ENTER)
```

A linha na parte inferior do vídeo deve estar mostrando

```
20 PRINT MANTEIGA, FERMENTO
```

Pressione **ENTER** e a linha 20 antiga será trocada pela atual. O vídeo agora deverá ficar assim:

```
10 LET MANTEIGA = 75
15 LET FERMENTO = 40
20  PRINT MANTEIGA, FERMENTO
```

Quando você executar o programa, ambos os preços serão impressos.

(Aqui está uma dica útil envolvendo **EDIT**, para ser usada quando você quiser limpar a parte inferior da tela. Pressione **EDIT** e a linha corrente será trazida para a parte inferior, cobrindo o que você quer cancelar. Se pressionar **ENTER**, a linha será recolocada no programa sem alterações, e a parte inferior do vídeo ficará limpa, deixando apenas o cursor.)

Agora digite:

```
12 LET FERMENTO = 40
```

Isso ficará no programa. Para cancelar essa linha desnecessária, digite

```
12          (com ENTER, é claro)
```

Você notará que o cursor desapareceu. Deve imaginá-lo como se ele estivesse entre as linhas 10 e 15, então se pressiona **↵** e ele irá para a linha 10 enquanto que, se pressionar **↵** ele irá para a linha 15.

Por último, digite

```
LIST 15
```

e você verá no vídeo

```
15  LET FERMENTO = 40
20 PRINT MANTEIGA, FERMENTO
```

A linha 10 desapareceu da tela, mas ainda se encontra no programa — o que você pode provar pressionando **ENTER** novamente. Os únicos efeitos de **LIST 15** são: fazer a listagem começar na linha 15 e colocar o cursor do programa na linha 15.

```
LIST
```

por si só, faz a listagem começar no início do programa.

#### Resumo

Programas  
Editando programas usando **↵**, **↵** e **EDIT**.  
Instruções: **RUN**, **LIST**

#### Exercícios

1. Modifique o programa de tal maneira que ele não só mostre os dois preços, mas também qual é qual.
2. Use a tecla **EDIT** para modificar o preço da manteiga.
3. Execute o programa e então digite

```
PRINT MANTEIGA, FERMENTO
```

As variáveis ainda estão lá, apesar do programa ter terminado.

4. Digite

12

(c **ENTER**)

Novamente, o cursor do programa ficará escondido entre as linhas 10 e 15. Agora pressione **EDIT** e a linha 15 virá para a parte inferior da tela: quando o cursor do programa está escondido entre duas linhas, **EDIT** começa na seguinte. Digite **ENTER** para limpar a parte inferior da tela.

Agora, digite

30

Dessa vez, o cursor do programa está escondido após o fim do programa; e se você pressionar **EDIT**, então a linha 20 virá para baixo.

5. Ponha uma instrução **LIST** no programa de tal maneira que, quando você executá-lo, ele liste a si mesmo.

# CAPÍTULO IX

## Mais programação

Digite

**NEW**

Isso apagará qualquer programa e variáveis do NE-Z8000. (Isto é quase como **CLEAR**, porém o **CLEAR** apaga somente as variáveis.)

Agora digite este programa cuidadosamente:

```
1Ø REM ESTE PROGRAMA EXTRAÍ RAIZ  
QUADRADA  
2Ø INPUT A  
3Ø PRINT A,SQR A  
4Ø GOTO 2Ø
```

**REM** na linha 1Ø é usado para comentários; está lá somente para lembrá-lo que o programa faz; o computador ignorará esta instrução.

Agora execute o programa. A tela aparentemente ficou branca e nada aconteceu; mas olhe no canto inferior esquerdo: onde deveria estar um **K** está um **■** — a máquina está esperando que seja digitado um número (ou até mesmo uma expressão), e ela não prosseguirá até que você tenha dado uma entrada de dados. Após isso, terá o mesmo efeito de

```
2Ø LET A = ... seja lá o que você digitou
```

Digite um número — 2, digamos — e então pressione **ENTER**. O 2 e sua raiz quadrada aparecerão na tela, e você pode pensar que só foi isso. Mas, não. A máquina quer outro número. Isso é devido à linha 4Ø, **GOTO 2Ø**, que significa “vá para a linha 2Ø”. Ao invés de executar e parar, o computador volta para a linha 2Ø e começa novamente. Assim, digite um outro número (é aconselhável que seja positivo).

Após algumas vezes; digite **STOP** (**SHIFT A**) ao invés do número.

Para reiniciar o programa, digite

**CONT**

(abreviatura de **CONTINUE**) e o computador limpará a tela e pedirá um outro número.

Para **CONT** o computador pega o número da linha na última denotação que tenha um código diferente de Ø e salta para aquela linha. Desde que a última denotação foi D/2Ø (e D não é Ø), em nosso caso **CONT** é idêntico a **GOTO 2Ø**.

Agora digite números até que a tela fique cheia. Quando encher, o computador parará com a denotação 5/3Ø. 5 significa “tela cheia”, e 3Ø é o número da instrução **PRINT** que estava tentando executar quando ele descobriu que não havia espaço. Novamente,

**CONT**

limpará a tela e continuará — dessa vez, **CONT** significa **GOTO 3Ø**.

Note que a tela é limpa, não porque **CONT** seja uma instrução, mas sim porque é um comando. Todos os comandos (exceto **COPY**, o qual aparece no capítulo 19) limpam primeiramente a tela.

Quando você estiver cansado disso, use **STOP** para parar o programa e obtenha uma listagem apertando **ENTER**.

Veja a instrução **PRINT** na linha 3Ø. Cada vez que o par de números **A** e **SQR A** são impressos, é numa nova linha, e isso porque a instrução **PRINT** não termina com vírgula ou ponto e vírgula. Sempre que isso ocorrer, então a próxima instrução **PRINT** começa a impressão em uma nova linha. (Assim, para imprimir uma linha em branco, use uma instrução **PRINT** na qual não haja nada para imprimir — somente um **PRINT** sozinho.)

Entretanto, uma instrução **PRINT** pode terminar em

uma vírgula ou ponto e vírgula, e então o próximo **PRINT** imprimirá como se os dois fossem uma longa instrução.

Por exemplo, com vírgulas, troque a linha 30 por

```
30 PRINT A,
```

e execute o programa para verificar quantas vezes sucessivas uma instrução **PRINT** pode imprimir na mesma linha, mas separados em duas colunas.

Com ponto e vírgula, por outro lado, com

```
30 PRINT A;
```

fica tudo espremido.

Tente também

```
30 PRINT A
```

Agora digite essas linhas extras:

```
100 REM ESSE PROGRAMA MEDE STRINGS
110 INPUT A$
120 PRINT A$,LEN A$
130 GOTO 110
```

Este programa é diferente do último, porém você pode manter ambos na memória da máquina ao mesmo tempo. Para executar o novo programa, digite

```
RUN 100
```

Neste programa é dada a entrada de uma string ao invés de números, imprimindo-as com seus tamanhos. Digite

```
GATO      (e ENTER, como sempre)
```

Devido ao computador estar esperando uma string, ele imprime duas aspas; isso é um alerta para você e, normalmente, evita também que você as digite. Mas você não tem que ficar restrito a constantes do tipo string (string explícita com aspas de abertura e fechamento e caracteres intermediários); o computador processará qualquer expressão string, tal qual uma com variáveis string. Nesse caso você teria que apagar as aspas impressas pelo computador. Tente isso. Remova (com  $\phi$  e **DELETE** duas vezes) e digite

```
A$
```

Desde que **A\$** ainda tenha o valor "GATO", o resultado é 4 novamente.

Agora dê entrada em

```
A$
```

novamente, dessa vez sem cancelar as aspas. Agora **A\$** tem o valor "A\$", e a resposta é 2.

Se desejar usar **STOP** para entrada de strings, deve primeiramente mover o cursor para o início da linha, usando  $\phi$ .

Agora olhe o comando **RUN 100** que demos anteriormente. Ele salta para a linha 100, não poderíamos, então, ter feito **GOTO 100**? Nesse caso a resposta é sim, mas há uma diferença. **RUN 100**, primeiramente, limpa as variáveis (como o **CLEAR** no capítulo 6), e após isso funciona da mesma forma que o **GOTO 100**. **GOTO 100** não limpa nada. Há certas ocasiões que se deseja executar o programa sem limpar as variáveis. Aqui **GOTO** é necessário e **RUN** poderia ser desastroso; assim é melhor não adquirir o hábito de digitar **RUN** automaticamente para executar um programa.

Uma outra diferença é que você pode digitar **RUN** sem número de linha, e ele iniciará a execução na 1ª linha do programa. **GOTO** sempre deve ter um número de linha.

Ambos os programas pararam porque você digitou **STOP** na linha de **INPUT**; porém, algumas vezes — devido a erro —, você escreve um programa que você não consegue parar e que não parará por si só. Digite

```
200 GOTO 200
RUN 200
```

Este parece que ficará eternamente em execução, a menos que você puxe a tomada; porém há uma solução menos drástica. Pressione a tecla **SPACE**, a qual, se você olhar bem, tem "BREAK" escrito em cima em letras, bem visíveis. O programa parará com denotação D/200.

Ao fim de cada linha do programa, o computador verifica se a tecla foi pressionada; se foi, então ela pára. O **BREAK** também pode ser usado quando estão sendo usados também o gravador e a impressora.

Agora você já viu as instruções **PRINT**, **LET**, **INPUT**, **RUN**, **LIST**, **GOTO**, **CONT**, **CLEAR**, **NEW** e **REM**, e a maioria delas pode ser usada tanto como comandos quanto como linhas de programa — o que é verdade para quase todas as instruções em **BASIC**. A única exceção, na realidade, é o **INPUT**, o qual não pode ser usado como um comando (você obterá uma denotação 8 se tentar; a razão é que a mesma área interna do computador é usada para comandos e área de dados, e para um comando **INPUT** haveria confusão). **RUN**, **LIST**, **CONT**, **CLEAR** e **NEW** não são de muita utilidade no programa, mas podem ser utilizados.

#### Resumo

Instruções: **GOTO**, **CONT**, **INPUT**, **NEW**, **REM**, **PRINT**, **STOP**, **BREAK**

#### Exercícios

1. No programa da raiz quadrada, verifique se trocar a linha 40 por **GOTO 5**, **GOTO 10** ou **GOTO 15** faz uma diferença perceptível na execução do programa. Se o número da linha de uma instrução **GOTO** se refere a uma linha inexistente, o salto será feito para a linha imediatamente posterior. O mesmo ocorrerá para **RUN**; na verdade, **RUN** sozinho significa **RUN 0**.

2. Execute o programa de tamanho de string; quando ele pede uma entrada de dados, digite

```
XS      (após remover as aspas)
```

É claro que XS é uma variável indefinida e você obterá uma denotação 2/11Ø.

Se você agora digitar

```
LET XS = "ALGUMA COISA DEFINIDA"
```

(a qual tem sua própria denotação Ø/Ø) e

```
CONT
```

você verá que pode usar XS como um dado de entrada sem nenhum problema.

O ponto chave nesse exercício é que **CONT** tem o mesmo efeito de **GOTO 11Ø**. Isso anula a denotação Ø/Ø, porque ela teve código Ø, e portanto pega o número da linha da denotação anterior, 2/11Ø. Isso é muito útil. Se um programa pára, devido a algum erro, você pode fazer qualquer tipo de coisa para corrigi-lo, e **CONT** ainda funcionará.

3. Tente este programa:

```
1Ø INPUT AS
2Ø PRINT AS;" = ";VAL AS
3Ø GOTO 1Ø
```

(veja capítulo 7, exercício 1).

Acrescente uma outra instrução **PRINT**, de tal maneira que o computador informe o que vai fazer e então peça os dados.

4. Escreva um programa para dar entrada de preços e fazer uma remarcação desses preços a uma taxa de (15%). Novamente, ponha uma instrução **PRINT**, de tal forma que o computador explique o que está fazendo. Modifique o programa para que possa também entrar a faixa de remarcação (para permitir orçamentos futuros).

5. Escreva um programa para imprimir o total dos números que digitar. (Sugestão: trabalhe com duas variáveis: **TOTAL** — que deve conter Ø no início — e **ITEM**. Dê entrada na variável **ITEM**, e adicione **TOTAL**, imprima ambos, e reinicie.)

6. As listagens automáticas (as que não são resultado de uma instrução **LIST**) podem chegar a confundir-lo. Se você digitar um programa com 5Ø linhas, todas com instruções **REM**,

```
1 REM
2 REM
3 REM
: :
: :
: :
49 REM
5Ø REM
```

você estará habilitado a experimentar.

A primeira coisa a lembrar é que a linha corrente (com **▣**) aparecerá na tela, preferivelmente próximo ao meio. Digite

```
LIST      (e ENTER, é claro)
```

então pressione **ENTER** novamente. Você deveria obter as linhas de 1 a 22 na tela. Agora digite

```
23 REM
```

Você deveria obter as linhas de 2 a 23 na tela. Digite

```
28 REM
```

e você obtém as linhas de 27 a 48. (Em ambos os casos, digitando uma nova linha, você move o cursor do programa de tal maneira que uma nova listagem deve ser feita.)

O computador mantém um registro não só de linha corrente — a que tem que aparecer na tela — mas também da linha do topo da tela. Quando ele tenta fazer uma listagem, a primeira coisa a ser feita é comparar a linha do topo com a linha corrente. Se a linha do topo vem após, não há razão de começar lá, assim ele usa a linha corrente para a nova linha do topo e faz a listagem.

Por outro lado, ele primeiramente tenta fazer a listagem começando na linha do topo. Se a linha corrente aparece na tela, tudo bem. Se a linha corrente está apenas um pouco abaixo da tela, ele move a linha do topo uma abaixo e tenta novamente; e se a linha corrente está bastante abaixo da tela, ele muda a linha do topo de tal maneira que ela se torne a linha anterior à linha corrente.

Experimente movendo a linha corrente pelo programa digitando

```
número da linha REM
```

**LIST** move a linha corrente, mas não a linha do topo; assim, listagens subsequentes devem ser diferentes. Por exemplo, digite

```
LIST
```

para obter a listagem, e então pressione **ENTER** novamente para fazer com que a linha Ø passe a ser a linha do topo. Você deveria ter as linhas 1 a 22 na tela. Digite

```
LIST 22
```

que fornece as linhas 2 a 43; quando você pressiona **ENTER** novamente, você obtém novamente as linhas 1 a 22. Isso tende a ser mais útil para pequenos programas do que para longos programas.

7. O que **CONT**, **CLEAR** e **NEW** fazem em um programa? Você pode pensar em algum uso para essas instruções?

# CAPÍTULO X

## IF...

Todos os programas vistos até agora têm sido bastante simples — eles seguiam todas as instruções e então voltavam às vezes ao início novamente. Isso não é muito útil. Na prática, o computador está apto a distinguir entre diferentes casos e agir de acordo; ele faz isso usando a instrução **IF**.

Limpe o computador (usando **NEW**), e digite e execute este ridículo programinha:

```
10 PRINT "POSSO CONTAR-LHE UMA
PIADA?"
20 INPUT A$
30 IF A$ = "DE JEITO NENHUM" THEN GO
TO 200
40 PRINT "QUANTAS PERNAS TEM UM
CAVALO MONTADO?"
50 INPUT PERNAS
60 IF PERNAS = 6 THEN GOTO 100
70 PRINT "NÃO, SEIS. QUATRO DO CAVA-
LO E DUAS DO CAVALEIRO"
80 STOP
100 PRINT "SIM", "QUER QUE EU A CON-
TE DE NOVO?"
110 GOTO 20
200 PRINT "ESTÁ BEM. NÃO CONTAREI"
```

Antes de discutirmos a instrução **IF**, você deve primeiro olhar a instrução na linha 80: uma instrução **STOP** para a execução do programa, dando uma indicação 9.

Agora, como você pode ver, uma instrução **IF** tem o seguinte formato:

**IF** condição **THEN** instrução

A instrução aqui é **GOTO**, mas poderia ser qualquer outra, até mesmo outro **IF**. A condição é alguma coisa que deve ser provada, ou verdadeira ou falsa. Se a condição for verdadeira, a instrução após o **THEN** é executada, caso contrário é ignorada.

A condição mais usada é comparar dois números ou duas strings: ela pode testar se dois números são iguais; ou se

um é maior que o outro; e pode testar se duas strings são iguais, ou se uma vem antes da outra em ordem alfabética. Para isso são usadas as relações =, <, >, <=, >= e <>.

=, o qual usamos duas vezes no programa (uma vez para números e outra para strings), significa "igual". Não é o mesmo que o = na instrução **LET**.

< significa "é menor que", assim

```
1<2
-2<-1
e -3<1
```

todos são verdadeiros, mas

```
1<0
e 0<-2
```

não (eles são falsos).

Para ver como isso funciona, vamos escrever um programa que pede para digitarmos uma série de números e imprime o maior deles até o momento.

```
10 PRINT "NÚMERO", "MAIOR ATÉ AGORA"
20 INPUT A
30 LET MAIOR = A
40 PRINT A, MAIOR
50 INPUT A
60 IF MAIOR < A THEN LET MAIOR = A
70 GOTO 40
```

A parte crucial está na linha 60, a qual atualiza **MAIOR** se seu valor antigo for menor que o novo valor digitado em **A**.

> (**SHIFT M**) significa "é maior que". É semelhante a <, só que é o inverso. Você pode distingui-los lembrando-se de que a parte mais aguda aponta para o número supostamente menor.

<= (**SHIFT R** — não digite como < seguido de =) significa "é menor que ou igual a". assim é igual a < exceto que

a decisão é tomada mesmo se os dois números são iguais: assim  $2 \leq 2$  executa, porém  $2 < 2$  não executa.

$> =$  (SHIFT Y) significa "é maior que ou igual a" e é similar a  $>$ .

$<>$  (SHIFT T) significa "não é igual a", o oposto em significado é  $=$ .

Matemáticos normalmente escrevem  $< =$ ,  $> =$  e  $<>$  como  $\leq$ ,  $\geq$  e  $\neq$ . Eles também escrevem seqüências como " $2 < 3 < 4$ " para significar " $2 < 3$  e  $3 < 4$ ", mas isso não é possível em BASIC.

Essas relações podem ser combinadas através do uso de operandos lógicos AND, OR e NOT.

uma relação AND outra relação

executa sempre que ambas forem verdadeiras.

uma relação OR outra relação

executa sempre que uma das duas relações for verdadeira.

NOT relação

executa sempre que a relação for falsa.

Expressões lógicas podem ser feitas com relações e AND, OR e NOT da mesma forma que as expressões numéricas podem utilizar números e +, - e assim por diante; você pode até mesmo colocar parênteses, se necessário, nas expressões lógicas. NOT tem prioridade 4, AND 3 e OR 2.

Para ilustrar, limpe o computador e tente esse programa.

```
10 INPUT F$
20 INPUT A
30 IF F$ = "X" AND A < 10 THEN
PRINT "SIM"
50 GOTO 10
```

Finalmente, podemos comparar não só números, mas também strings. Nós vimos "=" usado em  $F\$ = "X"$ . Você pode usar qualquer outro, como, por exemplo,  $<$ .

O que "menor que" significa para strings? Uma coisa que não significa é "menor que", assim não cometa este erro. Nós fazemos a distinção que uma string é menor que outra se ela vem primeiro em ordem alfabética: assim

"SMITH"	<	"SMYTHE"
"SMYTHE"	>	"SMITH"
"BILLION"	<	"MILLION"
"DOLLAR"	<	"POUND"

todas são verdadeiras.  $< =$  significa "é menor que ou igual a", e assim por diante, da mesma forma que para números

**Nota:** em algumas versões do BASIC — mas não no NE-Z8000 — a instrução IF pode ter o formato

IF condição THEN número da linha

Isso é o mesmo que

IF condição THEN GOTO número da linha

#### Resumo

Instruções: IF, STOP

Operações: =, <, >, < =, > =, <>, AND, OR

Função: NOT

#### Exercícios

1.  $<>$  e  $=$  são opostos no sentido que NOT  $A = B$  significa o mesmo que  $A <> B$  e

NOT  $A <> B$  é o mesmo que  $A = B$

Convença-se de que os pares,  $<$  e  $> =$ , e  $>$  e  $< =$  continuam da mesma maneira e você pode usar a instrução NOT em frente a uma das relações do par, para transformá-la na outra, sua negação.

Da mesma forma,

NOT (uma primeira expressão lógica, AND uma segunda expressão)

é o mesmo que

NOT (a primeira) OR NOT (a segunda),  
NOT (uma primeira expressão lógica OR uma segunda)

é o mesmo que

NOT (a primeira) AND NOT (a segunda).

Usando isso, você pode trabalhar com NOT entre parênteses, quando ele eventualmente for aplicado numa relação. Assim, você pode usá-lo à vontade. Entretanto, logicamente falando, NOT é desnecessário, porém você pode ainda achar que usando-o faz com que o programa torne-se mais claro.

2. No BASIC podem muitas vezes aparecer frases em inglês. Considere, por exemplo, a cláusula em inglês 'if A doesn't equal B or C'. Como você escreveria isso em BASIC? [A resposta não é

'IF A <> B OR C' nem 'IF A <> B OR A <> C']

Não se preocupe se você não entender os exercícios 3, 4 e 5, os pontos que são vistos neles são bastante refinados.

3. (A menos que você já conheça BASIC perfeitamente, salte esse exercício.)

Tente

PRINT 1 = 2, 1 <> 2

o que você espera que dê um erro de sintaxe. Na realidade, até onde o computador pode compreender, não existe nada além de valores lógicos.

(i) =, <, >, <=, >= e <> são todas operações binárias, com prioridade 5. O resultado é 1 (para verdadeiro), e 0 (para falso), quando o resultado de uma relação for 0 as instruções não são executadas.

(ii) em

**IF** condição **THEN** instrução

a condição pode, na realidade, ser qualquer expressão numérica. Se seu valor for zero, então é considerada como falsa e qualquer outro valor como verdadeiro. Assim, a instrução **IF** significa exatamente o mesmo que

**IF** condição <>0 **THEN** instrução

(iii) **AND**, **OR** e **NOT** são também operações binárias.

**X AND Y** tem valor  $\begin{cases} X \text{ se } Y \text{ não é zero} \\ (\text{é considerada como verdadeira}) \\ 0 \text{ se } Y \text{ é zero (é considerada} \\ \text{como falsa)} \end{cases}$

**X OR Y** tem valor  $\begin{cases} 1 \text{ se } Y \text{ não é zero} \\ X \text{ se } Y \text{ é zero} \end{cases}$

e **NOT X** tem valor  $\begin{cases} 0 \text{ se } X \text{ não é zero} \\ 1 \text{ se } X \text{ é zero} \end{cases}$

Com isso em mente, leia o capítulo novamente, certificando-se de que tudo isso funciona.

Nas expressões **X AND Y**, **X OR Y** e **NOT X**, X e Y irão novamente assumir o valor 0 ou 1, para falso ou verdadeiro. Idealize 10 condições diferentes e veja se nelas **AND**, **OR** e **NOT**, fazem o que você espera.

4. Tente este programa:

```
10 INPUT A
20 INPUT B
30 PRINT (A AND A > B) + (B AND A < B)
40 GOTO 10
```

Cada vez, ele imprime o maior número entre A e B. Por que?

Convença-se de que você pode pensar

**X AND Y**

como significando

'X se Y (ou então o resultado é 0)'

e

**X OR Y**

'X a menos que Y (no caso em que o resultado é 1)'

Uma expressão usando **AND** e **OR** dessa maneira é chamada de expressão **condicional**.

Um exemplo usando **OR** poderia ser

```
LET PREÇO AVULSO = PREÇO ATUAL*(1.15
OR US = "TAXA ZERO")
```

Agora note como **AND** tende à adição (porque seu valor de default é 0), e **OR** tende à multiplicação (porque seu valor de default é 1).

5. Você também pode fazer strings assumir valores em expressões condicionais, mas somente usando **AND**.

**X\$ AND Y** tem valores  $\begin{cases} X\$ \text{ se } Y \text{ não for zero} \\ "" \text{ se } Y \text{ for zero} \end{cases}$

então significa "X\$ se Y (ou então a string vazia)".

Tente este programa, que tem como entrada duas strings e colocando-as em ordem alfabética.

```
10 INPUT A$
20 INPUT B$
30 IF A$ < B$ THEN GOTO 70
40 LET C$ = A$
50 LET A$ = B$
60 LET B$ = C$
70 PRINT A$;" ";(" <" AND A$ < B$) +
(" = " AND A$ = B$);" "; B$
80 GOTO 10
```

6. Tente esse programa:

```
10 PRINT "X"
20 STOP
30 PRINT "Y"
```

Quando você executá-lo, ele irá imprimir "X" e parar com indicação 9/20. Agora digite

**CONT**

Você deve esperar que isso funcione como "GOTO 20". Dessa maneira, o computador iria apenas parar novamente, sem imprimir "Y". Isto não é muito útil, assim o programa é arranjado de tal maneira que, em indicações com código 9 (instrução **STOP** executada), o número da linha é acrescido de 1 para que a instrução **CONT** funcione corretamente. Assim, em nosso exemplo, "CONT" funciona como "GOTO 21" (o qual uma vez que não tem linhas entre 20 e 30, funciona como "GOTO 30").

7. Muitas versões do BASIC (mas não o BASIC do NE-Z8000) têm uma instrução **ON**, a qual tem o formato

**ON** expressão numérica **GOTO** número de linha, número de linha, ..., número de linha. Quando a expressão numérica é resolvida; suponha que o resultado obtido seja 3, então o efeito é o seguinte:

**GOTO** enésimo número da linha

Por exemplo,

**ON A GOTO 100, 200, 300, 400, 500**

Aqui, se o valor for 3, então 'GOTO 200' é executado. No BASIC do NE-Z8000 isso pode ser substituído por

**GOTO 100\*A**

Nos casos dos números das linhas não forem progredindo de 100 em 100, idealize uma maneira de utilizar esse recurso

**GOTO** uma expressão condicional

em vez do anterior.

# CAPÍTULO XI

## O conjunto de caracteres

As letras, dígitos, sinais de pontuação e outros sinais gráficos que podem aparecer em *strings* são chamados caracteres e formam o alfabeto, ou o conjunto de caracteres, que o NE-Z8000 usa. A maioria desses caracteres são símbolos simples, mas há alguns chamados **marca**, que representam uma palavra inteira, como no caso de **PRINT**, **STOP**, **\*\*** e outros.

Há 256 caracteres ao todo e cada um deles tem um código entre 0 e 255. Uma lista completa deles aparece no apêndice A. Para converter códigos e símbolos existem duas funções: **CODE** e **CHRS**

**CODE** é aplicada a uma string e fornece o código do primeiro caractere da string (ou 0, se a string for vazia)

**CHRS** é aplicada a um número e fornece o caractere cujo código é um número.

Esse programa imprime todo o conjunto de caracteres.

```
10 LET A = 0
15 SCROLL
20 PRINT CHRS A
25 PAUSE 20
30 LET A = A + 1
40 IF A < 256 THEN GO TO 20
```

No topo você pode ver os símbolos **^**, **£**, **\$** e assim por

diante, até Z; todos aparecem no teclado e podem ser digitados quando tivermos o cursor **█**. Mais à frente, você pode ver os mesmos caracteres em branco e preto (video inverso). Eles também podem ser obtidos do teclado. Se você pressionar **GRAPHICS (SHIFT 9)**, o cursor ficará **▣** significa **modo gráfico**. Se você digitar um símbolo, ele aparecerá na sua forma inversa, e isso continuará até que você pressione a tecla **GRAPHICS** novamente. **DELETE** terá seu significado normal. Tenha cuidado para não perder o cursor **▣** entre os caracteres inversos que você digitou.

Quando você tiver experimentado um pouco, você ainda deverá ter o conjunto de caracteres no topo da tela; caso contrário, execute o programa novamente. Os primeiros são um espaço em branco e dez símbolos gráficos de preto, branco e blocos cinzas. Adiante, há mais 11. Todos eles são chamados de **símbolos gráficos** e são utilizados para formar desenhos e figuras. Você pode digitá-los, pelo teclado, usando o modo gráfico. (Exceto para o espaço em branco, que é um símbolo comum, usando o cursor **█**; o quadrado preto é o espaço inverso.) Use as 20 teclas que têm símbolos gráficos escritos. Por exemplo: suponha que você desejasse o símbolo **■**, que está na tecla T. Pressione **GRAPHICS (SHIFT 9)** para obter o cursor **▣**; então pressione **SHIFT T**. Pelo que foi descrito anteriormente, você poderia estar esperando obter um inverso de símbolo; mas **SHIFT T** é normalmente **<>**, uma marca, e marcas não têm inverso. Então, ao invés, você consegue o símbolo gráfico **■**.

Aqui estão os 22 símbolos

Simbolo	Código	Como obter	Simbolo	Código	Como obter
	0	 SPACE		128	 SPACE
	1	 SHIFT 1		129	 SHIFT Q
	2	 SHIFT 2		130	 SHIFT W
	3	 SHIFT 7		131	 SHIFT 6
	4	 SHIFT 4		132	 SHIFT R
	5	 SHIFT 5		133	 SHIFT 8
	6	 SHIFT T		134	 SHIFT Y
	7	 SHIFT E		135	 SHIFT 3
	8	 SHIFT A		136	 SHIFT H
	9	 SHIFT D		137	 SHIFT G
	10	 SHIFT S		138	 SHIFT F

Agora veja o conjunto de caracteres novamente. As marcas aparecem bem claras em dois blocos. Há um pequeno grupo de 3 (RND, INKEY\$ e PI) após Z e um grande grupo, começando com as aspas após , e continuando de AT até COPY.

O restante dos caracteres parecem ser todos eles. Isso é na realidade apenas a maneira que eles são impressos; o ponto de interrogação real está entre : e (. Fora os espúrios, alguns são caracteres de controle, como , EDIT e ENTER, e o restante não tem nenhum significado especial para o NE-Z8000.

#### Resumo

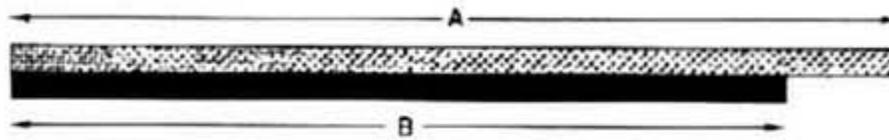
Funções: CODE, CHRS

#### Exercícios

1. Imagine o espaço para um símbolo dividido em quatro quartos:  Se cada quarto pode ser tanto preto como branco, há  $2 \cdot 2 \cdot 2 \cdot 2 = 16$  possibilidades. Encontre todas no conjunto de caracteres.

2. Imagine o espaço para um símbolo dividido em 2 horizontalmente:  Se cada metade pode ser branca, preta ou cinza, há  $3 \cdot 3$  possibilidades. Encontre-as.

3. Os caracteres do exercício anterior são desenhados para serem usados em histogramas horizontais, usando duas cores, cinza e preto. Escreva um programa que desenhe um histograma de dois valores A e B (ambos entre 0 e 32).



Você deverá iniciar imprimindo "█", então mudar ou para "██" ou para "████" dependendo se A for maior ou menor que B.

O que seu programa faz se A e B não são números inteiros? Ou eles não estão na faixa de 0 a 32? Um bom programa fará alguma coisa sensível e útil.

4. Há 2 caracteres todo cinza no teclado, em A e H. Se você olhá-los bem de perto, você verá que o H é uma miniatura de um tabuleiro de xadrez, enquanto que o A é um tabuleiro de xadrez invertido. Se você imprimi-los lado a lado verá que eles não se unem corretamente. O çõ A é usado para unir corretamente com ███ e ████ (em S e D), enquanto o do H une perfeitamente com ████ e ████ (em F e G).

5. Execute esse programa

```
1Ø INPUT A
2Ø PRINT CHR$ A;
3Ø GO TO 1Ø
```

Se você experimentar, você verá que CHR\$ é arredondado para o inteiro mais próximo. Se A não estiver na faixa de 0Ø até 255, o programa para com a denotação B.

6. Usando os códigos para caracteres, podemos expandir o conceito de ordem alfabética para ser aplicável a strings contendo qualquer caractere, não apenas letras. Se, ao invés de pensarmos em termos do alfabeto comum de 26 letras, nós utilizarmos o alfabeto expandido de 256 caracteres na mesma ordem de seus códigos, o princípio é exatamente o mesmo. Por exemplo, estas strings estão em ordem alfabética.

```
" ZACHARY"
" █ "
"(ASIDE)"
"123 TAXI SERVICE"
"AASVOGEL"
"AAHØVARK"
"ZACHARY"
"AA RDVARK"
```

Esta é a regra: primeiro, compare o 1º caractere nas duas strings; se eles forem diferentes, o código de um é menor que o código do outro, e a string que tem o primeiro caractere com código menor é a que está primeiro na ordem "alfabética". Se eles são iguais, prossiga e compare o segundo caractere. Se nesse processo, uma da strings acabar primeiro que a outra, esta é a anterior, caso contrário, elas são obviamente iguais.

Digite novamente o programa do exercício 4 do capítulo 1Ø aquele em que digitamos duas strings e ele os colocou em ordem e para experimentar.

7. Este programa imprime a tela com caracteres gráficos randomicamente.

```
1Ø LET A = INT (16*RND)
2Ø IF A >= 8 THEN LET A = A + 12Ø
3Ø PRINT CHR$ A;
35 PAUSE 2Ø
4Ø GO TO 1Ø
```

Como?

## CAPÍTULO XII

### Loops

Suponha que você quer somar cinco números. Uma maneira é escrever

```
1Ø LET TOTAL = Ø
2Ø INPUT A
3Ø LET TOTAL = TOTAL + A
4Ø INPUT A
5Ø LET TOTAL = TOTAL + A
6Ø INPUT A
7Ø LET TOTAL = TOTAL + A
8Ø INPUT A
9Ø LET TOTAL = TOTAL + A
10Ø INPUT A
11Ø LET TOTAL = TOTAL + A
12Ø PRINT TOTAL
```

Esse método não é nada bom para programar. Pode até ser controlável para cinco variáveis, mas imagine como seria monótono somar 1Ø números dessa maneira e 100 seria praticamente impossível. É muito melhor definir uma variável para contar até cinco e então parar o programa. Por exemplo:

```
1Ø LET TOTAL = Ø
2Ø LET COUNT = 1
3Ø INPUT A
4Ø REM COUNT = NÚMERO DE VEZES
QUE FOI DIGITADO
5Ø LET TOTAL = TOTAL + A
6Ø LET COUNT = COUNT + 1
7Ø IF COUNT <= 5 THEN GO TO 3Ø
8Ø PRINT TOTAL
```

Note como é fácil mudar a linha 7Ø para que o programa adicione 1Ø ou até 100 números.

Esse tipo de contagem é tão útil que a duas instruções para torná-la mais fácil: a instrução **FOR** e a instrução **NEXT**. Elas são sempre usadas juntas. Se você usar a instrução **FOR** e a instrução **NEXT** no programa anterior ele ficará assim:

```
1Ø LET TOTAL = Ø
2Ø FOR C = 1 TO 5
3Ø INPUT A
4Ø REM C = NÚMERO DE VEZES QUE FOI
DIGITADO
5Ø LET TOTAL = TOTAL + A
6Ø NEXT C
8Ø PRINT TOTAL
```

Para obter esse programa do anterior, você só tem que editar as linhas 2Ø, 4Ø, 6Ø e 7Ø. **TO** é **SHIFT 4**

Note que mudamos **COUNT** para **C**. A variável de contagem —ou variável de controle de um *loop* **FOR-NEXT**— deve ter apenas uma letra como nome.

O efeito desse programa é que **C** assume os valores 1 (o valor inicial), 2, 3, 4 e 5 (o limite); e para cada valor as linhas 3Ø, 4Ø e 5Ø são executadas. Então, quando **C** terminou seus 5 valores, a linha 8Ø é executada.

Um recurso extra é que a variável de controle não precisa ser iniciada com o valor 1: você pode mudar esse valor para qualquer outro, através do uso da parte **STEP** da instrução **FOR**. A forma mais geral para uma instrução **FOR** é:

**FOR** variável de controle e valor inicial **TO** limite **STEP** passo onde a variável de controle é apenas uma letra e

o valor inicial, limite e passo são todas expressões numéricas. Assim se você mudar a linha 20 do programa por:

```
20 FOR C=1 TO 5 STEP 3/2
```

C assumirá os valores 1, 2.5 e 4. Note que você não fica restrito a números inteiros, e também que o valor de controle não necessitará atingir o limite exatamente—ele permanecerá em looping enquanto seu valor for menor ou igual ao limite (mas veja o exercício 4).

Você deve tomar cuidado se estiver executando 2 loop **FOR-NEXT**, um dentro do outro. Tente esse programa, o qual imprime o conjunto completo de pedras de dominó:

```
10 FOR M=0 TO 6
20 FOR N=0 TO M
30 PRINT M;" ":" ";N;" "
40 NEXT N
50 PRINT
60 NEXT M
```

} Loop N } M

Você pode notar que o loop N está inteiramente contido no loop M—em outras palavras, eles estão em ninho. O que se deve evitar é dois loops **FOR = NEXT** sejam sobrepostos sem que estejam inteiramente um dentro do outro, como este:

```
5 REM PROGRAMA ERRADO
10 FOR M = 0 TO 6
20 FOR N = 0 TO M
30 PRINT M;" ":" ";N;" "
40 NEXT M
50 PRINT
60 NEXT N
```

} Loop M } N

### Resumo

#### Instruções: **FOR-NEXT**

Dois loops **FOR = NEXT** devem estar ou contidos um no outro ou completamente separados.

Outra coisa a evitar é pular de fora para dentro de um loop **FOR-NEXT**. A variável de controle é estabelecida corretamente apenas quando sua instrução **FOR** é executada, e se você omitir isso, a instrução **NEXT** causará problemas. Você deve conseguir uma denotação de erro 1 ou 2 (significando que uma instrução **NEXT** não contém uma variável de controle) se tiver sorte.

### Exercícios

1. Reescreva o programa do capítulo 11 que imprime o conjunto de caracteres, usando um loop **FOR-NEXT**

2. Uma variável de controle não tem apenas um nome e um valor, mas também um limite, um passo (incremento) e um número de linha para retorno do looping (a linha após a instrução **FOR**; convença-se que, primeiro, quando uma instrução **FOR** é executada, todas essas informações estão disponíveis (usando valor inicial como primeiro valor); e, segundo, que usando como um exemplo nosso segundo e terceiro programas) essa informação é suficiente para converter a linha

```
NEXT C
em duas linhas
LET C = C + 1
IF C <= 5 THEN GOTO 30
```

3. Execute o terceiro programa e digite

```
PRINT C
Por que a resposta é 6 e não 5?
```

4. Altere o programa para, ao invés de somar automaticamente 5 números, ele pergunte a quantidade de números que você quer somar. Quando você executar esse programa, o que acontece se você digitar 0, significando que você não quer somar nenhum número? Porque você esperaria que isso causasse problemas para o computador apesar de estar claro o que você deseja? (O computador tem que procurar a instrução **NEXT C**, o que nem sempre é necessário).

5. Tente este programa para imprimir números de 1 a 10 em ordem inversa.

```
10 FOR M = 10 TO 1 STEP -1
20 PRINT M
30 NEXT M
```

Converta este programa em um que não use o loop **FOR-NEXT** da mesma maneira que você converteria o program 3 no program 2 (veja exercício 1). Por que o passo negativo faz isso levemente diferente?

## CAPÍTULO XIII

### Sub-rotinas

Algumas vezes, partes diferentes de seu programa terão trabalhos similares a fazer, e você se encontrará digitando as mesmas linhas duas ou mais vezes. Entretanto, isto não é necessário. Você pode digitar as linhas uma única vez, na forma conhecida como **sub-rotina**, e, então, usá-las ou chamá-las em qualquer outro lugar do programa, sem ter que digitá-las novamente.

Para fazer isso use as instruções **GOSUB** e **RETURN**.

#### **GOSUB n**

Onde **n** é o número da primeira linha da sub-rotina, como **GO TO n**, exceto pelo fato do computador guardar o número da linha da instrução **GOSUB** para que ele possa retornar após a sub-rotina. Ele faz isso colocando o número da linha (o endereço de **retorno**) no topo de uma pilha de instruções (o **STACK GOSUB**)

#### **RETURN**

Pega o número da linha do topo do **stack GOSUB** e vai para a linha seguinte.

Como um primeiro exemplo:

```
10 PRINT "ESSE É O PROGRAMA PRIN-
CIPAL"
20 GOSUB 1000
30 PRINT "E NOVAMENTE"
40 GOSUB 1000
50 PRINT "E ISSO É TUDO"
60 STOP
1000 REM A SUB-ROTINA COMEÇA AQUI
1010 PRINT "ESSA É A SUB-ROTINA"
1020 RETURN
```

Sem a instrução **STOP** na linha 60 o programa correria para a sub-rotina e causaria erro 7 quando encontrasse a instrução **RETURN**.

Como outro exemplo, suponha que você deseja escrever um programa para lidar com jardas, pés e polegadas. Você terá 3 variáveis: **J**, **P** e **POL** (talvez outras-**J1**, **P1** e **POL1**, e assim por diante). A aritmética é fácil. Primeiro você separa as quantidades em jardas, pés e polegadas e trabalha com cada uma independentemente da outra- por exemplo, se você somar duas distâncias, você adiciona as polegadas, adiciona os pés e adiciona as jardas, separadamente. Para dobrar a distância, você dobra as polegadas, os pés e as jardas e então ajusta as quantidades para a forma correta, de maneira que as polegadas estejam entre 0 e 12 e os pés entre 0 e 3. O último estágio é comum a todas as operações, de forma que podemos fazê-lo em sub-rotina.

Deixando de lado a noção de sub-rotina por um momento, tente você mesmo fazer o programa, pois vale a pena. Dando valores arbitrários para **J**, **P** e **POL**, como convertê-los nos números corretos de jardas, pés e polegadas?

O que primeiro virá em mente é algo do tipo:  $1 J..4P..14 POL$ , que você deve converter em  $2J..2P..2POL$ . Isso não é muito difícil. Mas suponha que você tem números negativos. Voltemos a nossos valores iniciais- $1J..-4P..-14POL$ , que seria convertido para  $-2J..-2P..-2POL$ . E que tal frações? Se você dividir  $3J..1P..7 POL$  por dois você obtém  $1,5J..0,5P..3,5POL$ ; e apesar de ter pés entre 0 e 12, e jardas com 1,5, certamente não é tão bom quanto  $1J..2P..3,5POL$ .

Tente encontrar uma resposta para tudo isso e então use num programa-antes de seguir em frente com a leitura. Aqui está uma solução:

```

1000 REM SUB-ROTINA PARA AJUSTAR
J,P, POL PARA A FORMA CORRETA DE JARDAS,—,
PES, POLEGADAS

```

```

1010 LET POL = 36*J + 12*P + POL
1020 REM AGORA TUDO ESTÁ EM POLE-
GADAS
1030 LET E = SGN POL
1040 LET POL = ABS POL
1050 REM TRABALHAMOS COM POL POSI-
TIVO? MANTENDO SEU SINAL EM E
1060 LET P = INT(POL/12)
1070 LET POL = (POL-12*P)*E
1080 LET J = INT(P/3)*E
1090 LET P = P/E-3*J
1100 RETURN

```

Por si só isso não é de muita utilidade, porque não há programa para fazer alguma coisa com eles posteriormente. Digite o programa principal e também uma outra sub-rotina, para imprimir J, P e POL.

```

10 INPUT J
20 INPUT P
30 INPUT POL
40 GOSUB 2000
50 REM IMPRIME VALORES
55 PRINT TAB 8: "=";
60 GOSUB 1000
65 REM O AJUSTE
70 GOSUB 2000
75 REM IMPRIME OS VALORES
80 PRINT
90 GO TO 10
2000 REM SUB-ROTINA PARA IMPRIMIR J,
P, POL
2010 PRINT " ";J;"J."; P;"P."; POL;
"POL.."
2020 RETURN

```

Certamente, nós preservamos os valores usando a rotina de impressão na linha 2000, mas a sub-rotina de ajuste, na realidade, torna o programa mais longo — através de um **GOSUB** e **RETURN** — mas o tamanho do programa não é a única consideração. Bem usadas, as sub-rotinas podem tornar os programas mais fáceis de compreender.

O programa principal torna-se mais simples pelo fato de usar instruções mais poderosas: cada **GOSUB** representa algumas instruções BASIC complicadas. Mas você pode esquecer isso; apenas o resultado final importa. Graças a isso, é muito mais fácil seguir a estrutura principal do programa.

As sub-rotinas, por outro lado, são simplificadas por uma razão muito diferente, especificamente porque são mais curtas. Elas ainda usam as mesmas velhas e laboriosas instruções **LET** e **PRINT**, mas tem que fazer apenas parte do trabalho, e assim são mais fáceis de escrever.

O segredo consiste em escrever o nível-ou níveis- no qual escrever as sub-rotinas. Elas devem ser grandes o sufi-

ciente para ter um impacto significativo no programa principal, e ainda pequena o bastante para serem significativamente mais fáceis de escrever que o programa completo, sem sub-rotinas. Estes exemplos (não recomendados) ilustram bem. Primeiro:

```

10 GOSUB 1000
20 GOTO 10
1000 INPUT J
1010 INPUT P
1020 INPUT POL
1030 PRINT " "; J; "J."; P;"P.";POL,
"POL."; TAB 8: "=";
1040 LET POL = 36*J + 12*P + 1
: :
: :
2000 RETURN

```

E segundo:

```

10 GOSUB 1010
20 GOSUB 1020
30 GOSUB 1030
40 GOSUB 1040
5 GOSUB 1050
: :
: :
300 GO TO 10
1010 INPUT Y
1015 RETURN
1020 INPUT P
1025 RETURN
1030 INPUT POL
1035 RETURN
: :
: :

```

O primeiro, com sua única e poderosa sub-rotina e o segundo, com suas muitas sub-rotinas, demonstram extremos quase opostos, mas de igual futilidade.

Uma sub-rotina pode chamar outra, ou até mesmo ela própria (uma sub-rotina que chama si mesma é chamada de **recursiva**); portanto, não tenha receio de ter várias chamadas.

### Resumo

Instruções: **GOSUB**, **RETURN**

### Exercícios

1. O programa exemplo é virtualmente uma calculadora de distâncias universal. Como você o usaria

(I) Para converter Jardas e Polegadas em Jardas, Pés em Polegadas?

(II) Para converter metros em polegadas e pés?

(III) Para encontrar frações de uma Jarda? (ou seja, um terço de uma Jarda ou um pé.)

Inclua uma linha para arredondar polegadas para a polegada mais próxima.

2. Adicione duas instruções ao programa:

```
4 LET AJUSTE = 1000
7 LET JPOLIMP = 2000
Troque
GOSUB 1000 para GOSUB AJUSTE
GOSUB 2000 para GOSUB JPOLIMP
```

Isso funciona exatamente como você esperava. Na realidade, o número da linha em um **GOSUB** (ou **GOTO** ou **RUN**) pode ser qualquer expressão numérica (isto talvez não funcione para alguns computadores diferentes do NE-Z8000, porque ele não usa BASIC standard).

Esse tipo de coisa pode funcionar maravilhosamente bem para tornar seu programa mais claro.

3. Reescreva o programa principal do exemplo para fazer alguma outra coisa, mas usando a mesma sub-rotina.

```
4- GOSUB n
RETURN
em linhas consecutivas podem ser trocados por
GOTO n
Por que?
```

5. Uma sub-rotina pode ter vários pontos de entrada. Por exemplo, devido à forma como ele as usa, com **GOSUB 1000**, seguido imediatamente por **GOSUB 2000**, nós podemos substituir nossas duas sub-rotinas por uma grande que ajusta J, P e POL e os imprime. Ela tem dois pontos de en-

trada: um no início, para toda a sub-rotina e um outro mais adiante, para a parte de impressão apenas.

Faça os arranjos necessários.

6. Execute esse programa

```
10 GOSUB 20
20 GOSUB 10
```

Os endereços de retorno são colocados no Stack de **GOSUB** mas eles nunca são tirados; e eventualmente não há mais espaço para mais nada no computador. O programa para com indicação de erro 4. (Veja Apêndice B)

Você deve ter dificuldade em limpá-las sem perder tudo, mas uma dessas soluções funcionará:

(I) Cancele as duas instruções **GOSUB**

(II) Insira duas novas linhas

```
11 RETURN
21 RETURN
```

(III) Pressione

```
RETURN
```

Os endereços de retorno serão listados até você obter erro 7.

(IV) Altere seu programa para que não aconteça novamente. Como funciona?

# CAPÍTULO XIV

## Operando os programas

Existe bem mais na arte de programar computadores do que apenas saber o que faz cada instrução. Você já notou, provavelmente, que a maioria de seus programas apresentam o que é tecnicamente conhecido como *bug* (erro de programação, em inglês), sempre que você vai rodá-los pela primeira vez: às vezes são apenas erros de digitação, outras, erros em suas idéias do que o computador deveria fazer. Isso acontece normalmente devido à falta de uma maior experiência no assunto.

Fica então explicado esse ponto: quase todo programa, no início, apresenta *bugs* ou erros de programação.

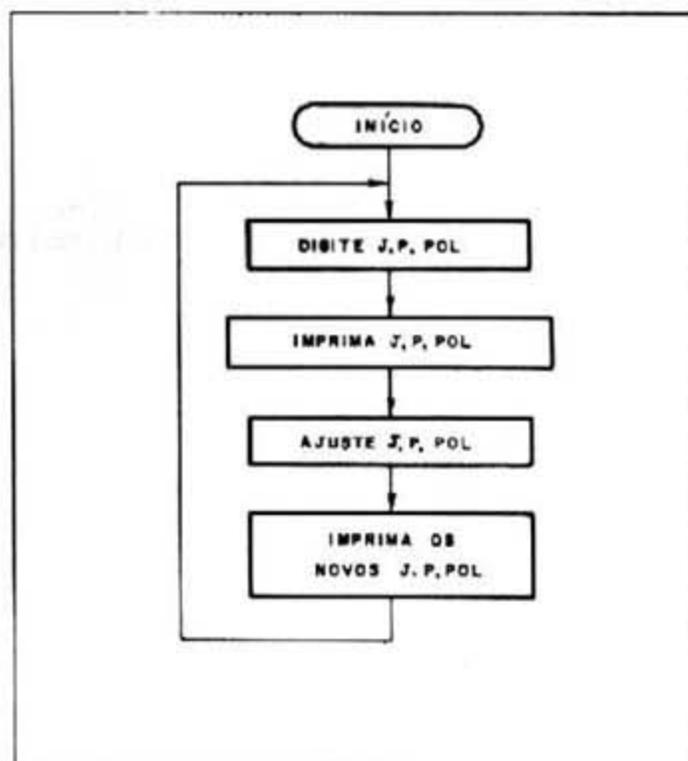
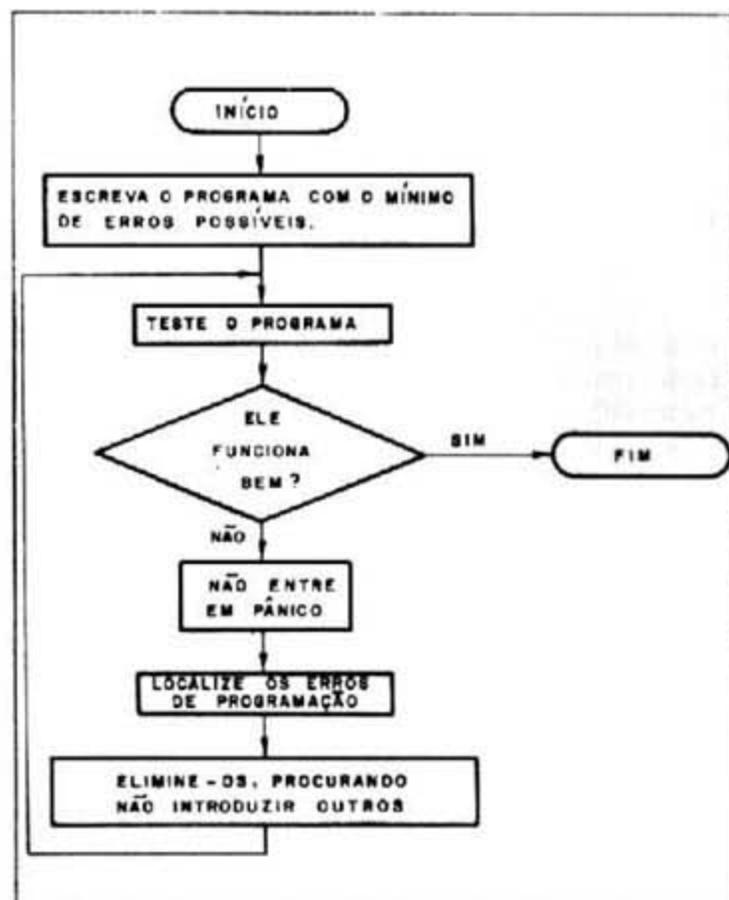
O plano geral de um programa pode ser facilmente ilustrados através de um **fluxograma**:

A idéia, aqui, é seguir os blocos de cima para baixo, de acordo com as setas, executando a instrução contida em cada uma delas. Costuma-se utilizar, como padrão, diferentes formatos de blocos para as várias instruções existentes. Assim: Um bloco arredondado  indica início ou fim. Um bloco retangular  indica uma instrução normal qualquer. Um losango  pede que se tome uma decisão, antes de prosseguir.

Esses formatos são amplamente utilizados, mas não são obrigatórios.

Os fluxogramas, naturalmente, servem para descrever a estrutura geral do programa, com uma sub-rotina em praticamente todos os blocos. Assim, o fluxograma para nosso exemplo de distâncias, no capítulo anterior, pode ser o seguinte:

Qualquer coisa — fluxogramas, sub-rotinas e também



as instruções **REM** — que torna o programa mais claro pode lhe fornecer uma melhor compreensão do mesmo; e, dessa forma, você pode certificar-se de cometer erros de programação em menor número. As sub-rotinas também ajudam a encontrar *bugs* já cometidos, tornando o programa mais fácil de testar. Você verá que é bem mais fácil testar cada sub-rotina individualmente e certificar-se de que elas se ajustam perfeitamente num todo, do que testar um programa inteiro não estruturado.

As sub-rotinas, portanto, auxiliam por meio do bloco “localize os erros de programação”, onde você poderá encontrar toda a ajuda que necessitar. Outras dicas para encontrar os famigerados *bugs* são as seguintes:

- 1) — Certifique-se sempre de que não há erros de digitação;
- 2) — Tente determinar o que todas as variáveis devem ser, a cada estágio — e, se possível, explique-as por meio de instruções **REM**. Você pode checar uma variável num determinado ponto de programa, inserindo uma instrução **PRINT** naquele ponto;
- 3) — Se um dos objetivos do programa é parar assim que surgir uma indicação de erro, utilize a informação o máximo que puder. Observe o código de erro e veja porque o progra-

ma parou numa determinada linha. Imprima os valores das variáveis, se necessário;

- 4) — Você deve estar apto a executar linha por linha do programa, digitando cada uma delas como comandos;
- 5) — Faça de conta que é o próprio computador e execute o programa, usando papel e lápis para anotar os valores das variáveis. Uma vez localizados os erros, corrigi-los é como escrever o programa original, mas é conveniente testar novamente o programa. É surpreendentemente fácil eliminar um *bug* e introduzir outro em seu lugar.

#### Exercícios

1. Digite um longo programa; depois, desligue o plugue da tomada. Isso é um tipo de coisa que acontece espontaneamente, algumas vezes; não é um erro de programação, e sim um problema elétrico e não há nada que você possa fazer a respeito. Caso isso aconteça com muita frequência, talvez haja algo errado; de qualquer forma, seria bom preservar os programas, mesmo incompletos, em fita magnética.
2. O fluxograma para o cálculo de distância não possui o bloco de “fim”; isso importa? Onde você o colocaria, caso desejasse?

## CAPÍTULO XV

# Armazenagem em fita

Como já foi mencionado no capítulo I — e você já não tem dúvidas graças a experiências anteriores — quando o NE-Z8000 é desligado, perde-se todo o programa e as variáveis que se encontravam armazenados na memória. A única maneira de preservá-los é instruir o computador a gravá-los numa fita cassette; desse modo, você poderá carregá-los de volta para a memória, posteriormente, e o computador estará praticamente no mesmo estado em que estava quando a gravação foi efetuada.

Junto ao computador você deve ter recebido um par de cabos, com os quais pode-se conectar o NE-Z8000 ao gravador. Convém você experimentar seu próprio gravador, pois alguns trabalham melhor que outros nessa função.

Em primeiro lugar, no que se refere ao computador, um gravador simples e barato tende a ser tão bom quanto os mais sofisticados, além de dar menos problemas, também.

Em segundo lugar, o gravador deve possuir uma tomada para microfone e outra para fone de ouvido. Devem ser, de preferência, do tipo *jack* fêmea, adequadas aos plugues fornecidos juntamente com os cabos.

Tendo então providenciado um gravador adequado, ligue-o ao NE-Z8000: um dos cabos deve interligar entrada de microfone e a tomada assinalada por "MIC", no computador. O outro cabo, por sua vez, deve interligar a saída de fone de ouvido à tomada "EAR" do NE-Z8000. Certifique-se de tê-los ligado corretamente.

Digite, então, um programa no computador — o programa, por exemplo, do conjunto de caracteres do capítulo II. Será preciso dar um nome ao programa, quando for armazená-lo, e seria uma boa idéia introduzir esse nome de maneira que ele apareça na listagem. A forma mais fácil é utilizar a instrução **REM**. Digite, então:

```
5 REM "CARACTERES"
```

Agora — e isto é apenas um exemplo, para que você possa visualizar melhor o que acontece — digite

```
SAVE "CARACTERES"
```

e observe a TV. Durante 5 segundos, ela ficará com a tela acinzentada; depois, por cerca de 6 segundos surgirá um desenho formado por listas brancas e pretas e, então, a tela ficará branca, com a indicação  $\theta\theta$ . O computador estava mandando um sinal para a tomada "MIC"; mas o mesmo sinal estava sendo enviado para a TV, produzindo a imagem que você viu. O período cinza era apenas silêncio, enquanto as faixas eram o programa.

Mas o que você quer fazer, obviamente, é capturar o sinal em fita; vamos fazer isso direito, desta vez.

### Preservando um programa

1. Posicione a fita num ponto em que esteja virgem ou possa ser regravada;
2. Usando um microfone, grave sua voz dizendo "caracteres". Isto não é essencial, mas tornará mais fácil a localização dos programas, posteriormente. Ligue novamente o computador ao gravador;
3. Digite, então

```
SAVE "CARACTERES" (sem ENTER)
```

4. Acione o gravador e dê início à gravação;
5. Pressione a tecla **ENTER**

6. Observe a TV, como anteriormente; quando a gravação terminar (com a indicação  $\emptyset\emptyset$ ), pare o gravador.

Para certificar-se de que tudo correu bem, você deve agora ouvir a fita gravada através do alto-falante do próprio gravador (provavelmente, você terá que desconectar o cabo do computador que está ligado à saída de fone de ouvido). Rebobine então a fita até o início do programa e ponha-a para "tocar".

Primeiramente, você ouvirá sua própria voz dizendo "caracteres". Depois, virá um zumbido suave; isso não é, na realidade, parte da gravação, mas o fim do sinal para a TV (antes que ENTER fosse pressionado), que também chegou ao gravador.

Em seguida, virão 5 segundos de silêncio, o início do sinal propriamente dito; corresponde ao período em que a tela tornou-se cinza. Depois, vem os 6 segundos de um zumbido estridente e alto, que a todo volume seria até desagradável; é a gravação do programa, correspondendo ao desenho em preto e branco visto na tela.

Por fim, retornará o zumbido suave, mais uma vez.

Caso você não ouça nada disso, certifique-se de que o computador e o gravador estão ligados corretamente. Em alguns modelos de gravador a tomada não faz contato se o plugue for totalmente introduzido; tente puxá-lo cerca de 2 mm para fora e você perceberá o encaixe numa posição mais natural.

Suponhamos agora que a gravação pareça estar correta ao ouvido e que você deseja carregá-la no computador.

#### Carregando um programa com nome

1. Rebobine a fita para o início do programa;
2. Certifique-se de que a tomada "EAR" do NE-Z8000 está perfeitamente ligada à tomada de fone de ouvido do gravador;
3. Gire o controle de volume do gravador até cerca de 3/4 do máximo; caso haja um controle de tonalidade, ajuste-o de modo mais agudo possível;
4. Digite

**LOAD "CARACTERES"**

(sem o ENTER, novamente)

5. Coloque o gravador em funcionamento;

6. Pressione o **ENTER**;

Mais uma vez, você verá o desenho da gravação na tela, mas um tanto diferente desta vez, em outra combinação de preto e branco. Será mais difícil distinguir entre o silêncio e o programa, mas você perceberá que a parte da programação apresenta linhas mais amplas e definidas (tente, se quiser, o exercício 1).

Após 15 segundos, aproximadamente, o programa deve estar carregado e encerrado com a indicação  $\emptyset\emptyset$ . Caso contrário, use a tecla **BREAK**.

Algo errado deve ter ocorrido, muito provavelmente, com o controle de volume. Ele deveria estar:

1. Alto o bastante para a parte de programa a ser captado pelo computador;

2. Não tão alto que levasse a distorcer o programa (o que é muito raro);

3. Baixo o suficiente para que a parte silenciosa fosse reconhecida pelo computador.

O melhor ajuste é girar o controle de volume sem que a parte silenciosa torne-se ruidosa; isto pode ser feito ao se ouvir a gravação pelo alto-falante. Caso o silêncio esteja realmente barulhento, surgirão alguns problemas:

— Alguns gravadores poderão formar um laço de realimentação com o NE-Z8000. Isso pode ser evitado desconectando o cabo "EAR", durante a gravação;

— Alguns gravadores — principalmente os mais antigos e usados — são intrinsecamente barulhentos. Uma fita de melhor qualidade pode ajudar nessa parte, apesar de não ser necessária, normalmente;

— Certos gravadores poderão captar o ruído de 60 Hz da rede. Resolva o problema alimentando-os com baterias ou pilhas;

— Por fim, o problema pode estar no plugue de fone de ouvido, que você empurrou completamente para dentro.

Essas regras devem englobar os problemas mais comuns; caso o defeito persista, desista e tente novamente na manhã seguinte.

Caso você tenha um programa em fita e não consegue lembrar seu título, ainda é possível carregá-lo (tente o exemplo com o programa "CARACTERES").

#### Carregando um programa sem nome

1. Posicione a fita na porção silenciosa;

2. Verifique tudo e ajuste os controles como anteriormente; você perceberá que o controle de volume deverá receber maiores cuidados que no caso anterior;

3. Digite

**LOAD ""**

(sem o ENTER)

4. Ponha o gravador para funcionar;

5. Pressione o **ENTER**;

6. O restante procede como já foi explicado.

A idéia, aqui, é que se o nome do programa que você pede para carregar é um *string* vazia, o computador carrega o primeiro programa que encontra pela frente. Note que, quando você preserva um programa, você não pode fazer uma *string* vazia de seu nome. Caso você tente, mesmo assim, obterá o sinal de erro F.

**LOAD** e **SAVE** também podem ser usadas em programas. Com **SAVE**, o programa se auto-preservará de tal forma que, quando for carregado, ele continuará rodando imediatamente. Digite, a título de exemplo:

**5 REM "INÚTIL"**

**10 PRINT "ISTO É TUDO O QUE ELE FAZ"**

**20 STOP**

**100 SAVE "INÚTIL"**

**110 GOTO 10**

Ligue o gravador e digite

**RUN 100** (sem o ENTER)

ponha o gravador para funcionar e pressione **ENTER**. Quando o programa tiver se preservado, continuará a rodar normalmente.

Para carregar esse programa, agora, rebobine a fita até seu início e digite

**LOAD "INUTII"** (sem o ENTER)

ponha o gravador para funcionar e pressione **ENTER**. Assim que o programa estiver carregado, ele irá para a linha 100 e prosseguirá, sem qualquer esforço de sua parte.

Note que colocar a instrução **SAVE** no final de um programa significa que, para executá-lo sem o **SAVE**, você tem apenas que digitar **RUN**, podendo omitir o próprio **SAVE**.

Não utilize o **SAVE** numa rotina com **GOSUB** — ele não funcionará.

Não inclua caracteres de vídeo inverso no nome de um programa; tudo o que houver após o caractere perde-se. E o nome não deve conter mais que 127 caracteres.

O nome em um **LOAD** ou **SAVE** não precisa ser uma constante *string*; pode ser qualquer expressão *string*, como **A\$** ou **CHR\$ 100**.

#### Resumo

- Preservando um programa em fita
- Carregando um programa com nome
- Carregando o primeiro programa disponível de uma fita
- Preservando um programa que possa ser carregado e continue rodando

Instruções: **SAVE**, **LOAD**

**Nota:** Não é possível carregar programas que foram preservados em outros computadores ou no NE-Z80, com seu BASIC próprio. Da mesma forma os programas desenvolvidos no NE-Z8000 não podem ser transferidos para outros computadores ou para o NE-Z80.

#### Exercícios

1. Faça uma fita com vários programas pequenos, comece a carregá-los no computador e digite

**LOAD "NÃO O NOME DE UM PROGRAMA"**

Você perceberá facilmente a diferença, na tela, entre os espaços vazios da fita (com uma imagem desestruturada) e os programas (com linhas mais definidas). Ambas as imagens são diferentes da que você vê quando preserva um programa. Caso o volume seja baixado durante a passagem de um programa, você poderá ver a imagem mudando para a forma de vazio, assim que o sinal se torna muito fraco para ser identificado como programa.

2. Faça uma fita na qual o primeiro programa, quando carregado, imprime uma lista dos demais programas da fita, pede para que um deles seja escolhido e o carrega.

3. Digite novamente o programa "CARACTERES" e então digite

**LET X = 7**

de maneira que — apesar da mesma não aparecer no programa — o computador agora contenha uma variável X de valor 7. Preserve então o programa, desligue o computador, ligue-o novamente e carregue o programa mais uma vez. Digite

**PRINT X**

e você obterá a resposta 7. A instrução **SAVE** preservou não só o programa, mas todas as variáveis, incluindo X. Caso você queira manter as variáveis quando executa o programa, deve lembrar-se de usar **GOTO** e não **RUN** (como foi mencionado no capítulo 9). Para não ter que lembrar-se disso toda vez que for rodar um programa, faça os programas se auto-executarem (empregando **SAVE** como linha de programação).

4. Digite um longo programa e desligue momentaneamente a alimentação do computador. Como dissemos, esse tipo de coisa costuma acontecer espontaneamente na rede elétrica; não é um erro mas um transiente. Mais uma vez, repetimos: se ocorrer com muita frequência, algo deve estar errado, mas será conveniente ir preservando programas longos aos poucos, em fita.

## CAPÍTULO XVI

### Imprimindo mais esteticamente

Você deve se lembrar que uma instrução **PRINT** possui uma lista de itens, sendo cada um deles uma expressão (ou talvez nada de significativo), e que são separados por vírgula ou ponto-e-vírgula. Existem mais dois itens **PRINT**, usados para dizer ao computador não o que, mas onde imprimir. Assim, por exemplo, **PRINT AT 11, 16; "\*"'** imprime um asterisco bem no meio da tela.

Desse modo, **AT** linha, coluna move a posição **PRINT** (ou seja, o lugar onde o próximo item deve ser impresso) para a linha e coluna especificadas. As linhas são numeradas de 0 (em cima) a 21, enquanto as colunas são numeradas de 0 (à esquerda) a 31.

Da mesma forma, **TAB** coluna move a posição de **PRINT** para a coluna especificada. Permanece, porém, na mesma linha ou, caso envolva retorno, transfere-se para a linha de baixo.

Observe que o computador reduz o número da coluna módulo 32 (divide por 32 e toma o resto); assim **TAB 33** significa o mesmo que **TAB 1**.

Por exemplo, vamos imprimir o cabeçalho de uma página chamado "Conteúdo", sendo 1 o número dessa página:

```
PRINT TAB 30;1;TAB 12; "CONTEUDO";  
TAB 24; "PÁGINA"
```

Alguns pontos a considerar:

1. Esses dois novos itens tem, como melhor terminação, o ponto-e-vírgula, como fizemos acima. Mas você pode usar vírgula (ou nada) ao final da instrução; porém, isso vai significar que, após você ter definido cuidadosamente a posição do **PRINT**, ele irá se mover novamente;

2. Apesar de **AT** e **TAB** não serem funções, é preciso pressionar a tecla **FUNCTION (SHIFT ENTER)** para obtê-las;

3. Não é possível imprimir nas duas linhas inferiores (22 e 23) da tela. Quando nos referimos à linha mais baixa, é da 21 que estamos falando;

4. Você pode usar **AT** para posicionar o **PRINT** até mesmo onde já existe algo impresso; a inscrição existente será apagada.

Há mais duas instruções ligadas ao **PRINT**, denominadas **CLS** e **SCROLL**. A primeira limpa a tela, enquanto a segunda move a imagem toda uma linha para cima (perdendo-se, então, a linha do topo) e desloca a posição **PRINT** para o início da linha inferior (linha 21).

Para ver como ela funciona, vejamos este programa:

```
10 SCROLL  
20 INPUT A$  
30 PRINT A$  
40 GOTO 10
```

#### Resumo

Itens tipo **PRINT**: **AT**, **TAB**

Instruções: **CLS**, **SCROLL**

#### Exercícios

1. Tente rodar o seguinte programa:

```
10 FOR I = 0 TO 20  
20 PRINT TAB 8*I;I;  
30 NEXT I
```

Ele mostra o que significa quando o número **TAB** é reduzido para módulo 32. Para obter um exemplo mais interessante, troque o número 8, na linha 20, por um 6.

## CAPÍTULO XVII

## Gráficos

Aqui está uma das mais atraentes características do NE-Z8000, utilizando os elementos de imagem. A tela usada como *display* para o computador conta com 22 linhas e 32 colunas, formando  $22 \times 32 = 704$  posições de caractere, cada uma contendo 4 elementos de imagem.

O elemento de imagem é especificado por dois números, que são suas coordenadas. O primeiro, sua coordenada *x*, determina a distância que ele se encontra da coluna mais à esquerda; e o segundo, a coordenada *y*, diz o quão elevado ele está em relação à linha mais baixa. Tais coordenadas são normalmente escritas como um par de números entre parênteses, de forma que (0,0), (63,0), (0,43) e (63,43) correspondem, respectivamente, aos cantos inferior esquerdo, inferior direito, superior esquerdo e superior direito da tela.

A instrução

**PLOT** coordenada *x*, coordenada *y*

faz com que o elemento de imagem, em preto, coincida com essas coordenadas, enquanto

**UNPLOT** coordenada *x*, coordenada *y*

o apaga.

Tente este programa simples:

```
10 PLOT INT (RND*64), INT(RND*44)
20 INPUT AS
30 GOTO 10
```

Esse programa plota um ponto aleatório cada vez que o **ENTER** é pressionado. Eis um programa muito mais útil; ele plota o gráfico da função seno (ou seja, uma senóide) para valores entre 0 e  $2\pi$ .

```
10 FOR N=0 TO 63
20 PLOT N,22 + 20*SIN(N/32*PI)
30 NEXT N
```

Este outro monta o gráfico da função **SQR** (parte de uma parábola) entre 0 e 4:

```
10 FOR N=0 TO 63
20 PLOT N, 20*SQR(N/16)
30 NEXT N
```

Note que as coordenadas do elemento de imagem são bastante diferentes das linhas e colunas em um item **AT**. Você vai perceber como é útil o diagrama final deste capítulo para se trabalhar com coordenadas dos elementos de imagem e com os números de colunas e linhas.

### Exercícios

1. Há três diferenças entre os números de um item **AT** e as coordenadas do elemento de imagem. Quais são elas?

Suponha que a posição do **PRINT** corresponda a **AT** *L*, *C* (linha e coluna). Prove para si mesmo que os 4 elementos de imagem daquela posição possuem coordenadas  $x \cdot 2^*C$  ou  $2^*C + 1$  e coordenada  $y \cdot 2^*(21-L)$  ou  $2^*(21-L) + 1$  (veja o diagrama).

2. Procure alterar aquele programa simples, de modo que ele preencha, primeiramente, a tela de preto (um quadrado preto é um espaço de vídeo inverso), e então utilize a instrução **UNPLOT** sobre alguns pontos aleatórios. Caso você possua apenas 1 k de RAM — ou seja, o computador padrão sem a expansão de memória — você não terá espaço suficiente e precisará alterar o programa, de forma a utilizar somente uma parte da tela.

3. Modifique o programa do gráfico senoidal, a fim de que ele imprima, antes de plotar o gráfico, uma linha horizontal de vários "-" como eixo x e vários "/" para o eixo y.

4. Escreva programas para montar gráficos de outras funções, tais como **COS**, **EXP**, **LN**, **ATN**, **INT** e assim por diante. A cada gráfico, você deve certificar-se de que o mesmo cabe na tela, considerando:

- Em que faixa de valores você tomará a função (correspondendo à faixa entre  $0$  e  $2\pi$  para o seno);
- Onde colocar o eixo x na tela (correspondendo a 22 na linha 20 do programa do seno);
- Como determinar a escala do eixo y do gráfico (correspondendo a 20 na linha 20 do programa do seno).

5. Rode este programa:

```
10 PLOT 21, 21
20 PRINT "ASPAS"
30 PLOT 46, 21
```

**PLOT** desloca a posição **PRINT** e **UNPLOT** também.

6. Esta sub-rotina traça uma linha quase reta do elemento de imagem (A, B) até (C, D). Use-a como parte de um programa principal que forneça os valores A, B, C e D (caso você não possua a expansão de memória, talvez tenha que omitir as instruções **REM**).

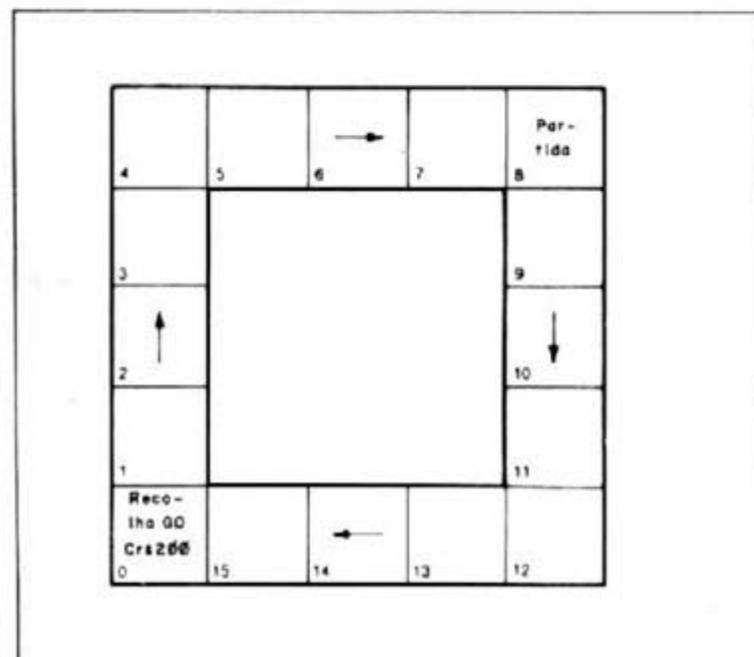
```
1000 LET U=C-A
1005 REM U MOSTRA QUANTOS PASSOS
TEMOS QUE DAR
1010 LET V=D-B
1015 REM V MOSTRA QUANTOS PASSOS
PARA CIMA
1020 LET D1X=SGN U
1030 LET D1Y=SGN V
1035 REM (D1X,D1Y) É UM ÚNICO PASSO
EM DIAGONAL
1040 LET D2X=SGN U
1050 LET D2Y=0
1055 REM (D2X,D2Y) É UM ÚNICO PASSO À
DIREITA OU ESQUERDA
1060 LET M=ABS U
1070 LET N=ABS V
1080 IF M>N THEN GOTO 1130
1090 LET D2X=0
1100 LET D2Y=SGN V
1105 REM AGORA (D2X,D2Y) É UM ÚNICO
PASSO PARA CIMA OU PARA BAIXO
1110 LET M=ABS V
1120 LET N=ABS U
1130 REM M É O MAIOR DE ABS U E ABS V,
N É O MENOR
1140 LET S=INT(M/2)
1145 REM QUEREMOS NOS DESLOCAR DE
(A,B) PARA (C,D) EM M PASSOS USANDO N PASSOS
```

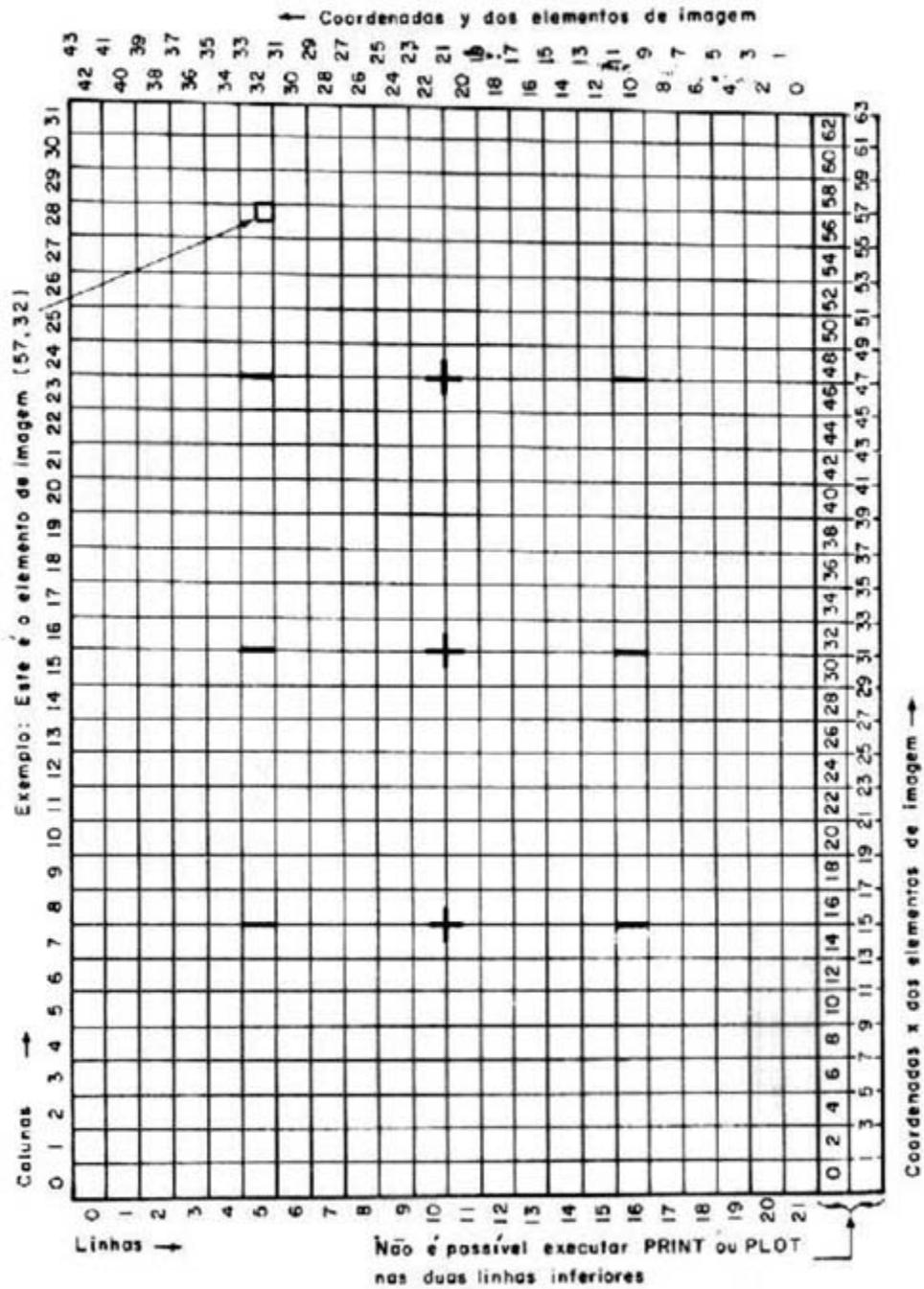
PARA CIMA OU PARA BAIXO OU D2 PASSOS PARA A DIREITA OU ESQUERDA E D1 PASSOS DIAGONAIS M—N, DISTRIBUÍDOS O MAIS UNIFORMEMENTE POSSÍVEL.

```
1150 FOR N=0 TO M
1160 PLOT A,B
1170 LET S=S+N
1180 IF S<M THEN GOTO 1230
1190 LET S=S-M
1200 LET A=A+D1X
1210 LET B=B+D1Y
1215 REM UM PASSO EM DIAGONAL
1220 GOTO 1250
1230 LET A=A+D2X
1240 LET B=B+D2Y
1245 REM UM PASSO PARA CIMA/BAIXO
OU PARA A DIREITA/ESQUERDA
1250 NEXT N
1260 RETURN
```

A última parte (da linha 1150 em diante) mistura uniformemente os passos D1 M—N com os N passos D2. Imagine um tabuleiro de jogo Monopólio com M quadrados em toda a volta, numerados de 0 até M—1; o quadrado no qual você está em qualquer momento é o de número S, começando no canto oposto ao do **GO**. Cada movimento o leva N quadrados ao longo do tabuleiro e em linha reta, na tela, você faz um movimento vertical ou horizontal (caso você passe pelo **GO** no tabuleiro), ou então um passo em diagonal. Como seu deslocamento total no tabuleiro é de M\*N passos, ou todo o percurso N vezes, você passa por **GO** N vezes também, e igualmente espaçados em seus M passos encontram-se N passos esquerda/direita ou acima/abaixo.

Ajuste o programa de forma que se outro parâmetro — E — seja 1, a linha seja traçada em preto (como aqui) e caso seja 0, a linha é traçada em branco (usando **UNPLOT**). Você pode, então, cancelar uma linha que já havia traçado.





## CAPÍTULO XVIII

# Tempo e movimento

Freqüentemente você desejará fazer um programa que dure um tempo específico na tela, e para esse propósito você achará a instrução **PAUSE** útil.

### **PAUSE n**

para durante n quadros da televisão (com 60 quadros por segundo); n pode ser usado até 32767, o que dá cerca de 11 minutos; se n é maior, significa que deve parar para sempre.

Ao fim de uma pausa, a tela brilhará mais forte.

A pausa pode ser abreviada pelo acionamento de uma tecla, após o início do período de pausa. A instrução **PAUSE** deve ser seguida por **POKE 16437, 255**.

A **PAUSE** aparentemente funciona sem isso, mas às vezes poderá abortar seu programa. Este programa simula o ponteiro dos segundos de um relógio:

```
5 REM PRIMEIRO DESENHAMOS O RE-
LÓGIO
10 FOR N=1 TO 12
20 PRINT AT 10-10*COS(N/6*PI),10+10*
SIN(N/6*PI); N
30 NEXT N
35 REM AGORA COMEÇANDO O FUNCIO-
NAMENTO DO RELÓGIO
40 FOR T=0 TO 1000
45 REM T É O TEMPO EM SEGUNDOS
50 LET A = T/30*PI
60 LET SX = 21 + 18 * SIN A
70 LET SY = 22 + 18 * COS A
200 PLOT SX, SY
300 PAUSE 42
305 POKE 16437, 255
310 UNPLOT SX, SY
400 NEXT T
```

(Deixe de lado as instruções **REM**, a menos que você tenha uma placa de expansão de memória.)

Esse relógio parará após 2:45 devido a linha 40, mas você pode facilmente fazê-lo funcionar mais tempo. Note como o tempo é controlado pela linha 300. Você esperaria **PAUSE 60** para fazê-lo mudar uma vez por segundo, mas o processamento também toma tempo e esse tempo tem que ser considerado. A melhor maneira de fazê-lo é através de tentativa e erro, cronometrando o relógio do computador com um real e ajustando a linha 300. (Você não pode fazer isso com precisão; um ajuste de um quadro por segundo é 2% ou meia hora em um dia).

A função **INKEY\$** (a qual não tem argumento) lê o teclado como se você estivesse pressionando uma tecla; o resultado é caractere que aquela tecla dá em modo **U**; caso contrário, o resultado é uma *string* vazia. Os caracteres de controle não tem o efeito comum, porém dão resultado como **CHR\$ 118** para **ENTER** — eles são impressos como“?”.

Tente este programa, que funciona como uma máquina de escrever.

```
10 IF INKEY$ <> "" THEN GOTO 10
20 IF INKEY$ = "" THEN GOTO 20
30 PRINT INKEY$;
35 PAUSE 20
40 GOTO 10
```

A linha 10 espera você retirar seu dedo do teclado. E linha 20 espera você pressionar uma tecla.

Lembre-se que ao contrário de **INPUT**, **INKEY\$** não espera você. Assim você não digita **ENTER**. Por outro lado, se você não digitar nada, você terá perdido a sua chance.

### Exercícios

1. O que acontece se você retirar a linha 10 do programa da máquina de escrever?
2. Por que você não pode digitar espaço ou  $\text{␣}$  no programa da máquina de escrever?

Aqui está uma modificação que fornece um espaço se você digitar "CURSOR PARA A DIREITA" (SHIFT 8)

```
10 IF INKEYS <>" " THEN GOTO 10
20 IF INKEYS = " " THEN GOTO 20
30 LET AS = INKEYS
40 IF AS = CHR$ 115 THEN GOTO 110
90 PRINT AS;
100 GOTO 10
110 PRINT " ";
120 GOTO 10
```

Note que transportamos **INKEYS** para **AS** na linha 30. Seria possível omitir isso e repor **AS** por **INKEYS** nas linhas 40 e 90 mas haveria sempre uma chance de **INKEYS** mudar entre as linhas.

Acrescente alguma coisa mais no programa, de tal maneira que, se você digitar **ENTER** (CHR\$ 118), ele passe para uma nova linha.

3. Você ainda pode usar **INKEYS** em conjunto com **PAUSE** como nesse programa alternativo de máquina de escrever.

```
10 PAUSE 40000
20 POKE 16437, 255
30 PRINT INKEYS;
40 GOTO 10
```

Para fazer esse programa funcionar, porque é essencial que uma pausa não termine se encontrar você pressionando a tecla quando ele inicia?

Esse método tem a desvantagem da tela piscar. Note que o computador aproveita a oportunidade de uma pausa para imprimir a imagem na televisão.

4. O programa seguinte faz o computador imprimir um número, o qual você (ou uma vítima inocente) deveria digitar. Para começar, você tem um segundo para fazê-lo, mas se fizer errado você tem maior tempo para o próximo número,

entretanto se você acertar, terá menos tempo para a próxima vez. O ideal é fazê-lo o mais rápido possível. E então pressionar Q para ver sua marcação de pontos — quanto maior, melhor.

```
10 LET T = 50
15 REM T = NÚMERO DE CICLOS POR JOGADA — INICIALMENTE 50 para 1 segundo
20 SCROLL
30 LET AS = CHR$ INT (RND * 10 + CODE
"0")
35 REM AS É UM DÍGITO RANDÔMICO
45 PAUSE T
50 POKE 16437, 255
60 LET BS = INKEYS
70 IF BS = "Q" THEN GOTO 200
80 IF AS = BS THEN GOTO 150
85 SCROLL
90 PRINT "NADA BOM"
100 LET T = T * 1.1
110 GOTO 20
115 SCROLL
150 PRINT "OK"
160 LET T = T * 0.9
170 GOTO 20
200 SCROLL
210 PRINT "VOCÊ FEZ"; INT (500 / T);
"PONTOS"
```

5. (APENAS PARA AQUELES COM EXPANSÃO DE MEMÓRIA)

Usando a rotina da linha reta do capítulo XV, mude o programa do relógio para que ele também mostre ponteiros de minutos e horas (faça o ponteiro de horas menor), arranje para que ele faça um tipo de marcação a cada quarto de hora.

# CAPÍTULO XIX

## A impressora

Este capítulo é sobre instruções BASIC para o uso de impressora.

As duas primeiras, **LPRINT** e **LLIST**, são como **PRINT** e **LIST**, exceto por usarem a impressora ao invés da televisão. (o L é um acidente histórico. Quando o BASIC foi inventado, ela geralmente usava uma máquina de escrever elétrica ao invés de uma televisão, assim **PRINT** realmente significava imprimir. Se você necessitasse de muita saída de dados usaria uma impressora de linha acoplada ao computador, e **PRINT** significava "imprimir na impressora de linha").

Ente este programa, por exemplo:

```
10 LPRINT "ESTE PROGRAMA"....
20 LLIST
30 LPRINT "IMPRESSÃO DO CONJUNTO
DE CARACTERES" ...
40 FOR N = 0 TO 255
50 LPRINT CHR$ N;
60 NEXT N
```

A terceira instrução, — **COPY**, imprime uma cópia da tela da televisão. Por exemplo, obtenha uma listagem do programa acima na tela, e digite:

### COPY

Você pode sempre parar a impressora quando ela está em funcionamento pressionando a tecla **BREAK** (espaço).

### Resumo

Instruções: **LPRINT**, **LLIST**, **COPY**

Nota: nenhuma dessas instruções são padrão em BASIC, apesar de que **LPRINT** ser usada por outros computadores.

### Exercícios:

1. Ente este:

```
10 FOR N = 31 TO 0 STEP - 1
20 PRINT AT 31 - N, N; CHR$ (CODE "0" +
N);
```

### 30 NEXT N

Você verá um desenho de letras em diagonal desde do canto superior direito até o canto inferior esquerdo da tela, quando o programa parar com denotação de erro (5/20).

Agora mude "**AT 31—N,N**" na linha 20 por "**TAB N**". O programa fará o mesmo que anteriormente.

Agora mude **PRINT** na linha 20 por **L PRINT**. Dessa vez não haverá erro 5, o qual não deve ocorrer com a impressora, e o desenho continuará por mais 10 linhas com os dígitos.

Agora mude "**TAB N**" por "**AT 21—N,N**", ainda usando símbolos. A razão da diferença é que a saída do **L PRINT** não é impressa imediatamente, mas colocada num *buffer* de armazenamento, a imagem de uma linha de impressão que o computador imprimirá de uma só vez. A impressão ocorrerá:

- (I) Quando o **BUFFER** estiver completo
- (II) após uma instrução que não termina em vírgula ou ponto e vírgula,
- (III) Quando uma vírgula ou um item **TAB** requer uma nova linha, ou
- (IV) no fim de um programa, se houver algo que não foi impresso.

O número (III) explica porque nosso programa com **TAB** funciona dessa maneira. Como para **AT**, o número da linha é ignorado e a posição do **LPRINT** (como a posição do **PRINT**, porém para a impressora ao invés da televisão) é mudado para número da coluna um item, **AT** nunca manda uma linha para a impressora (na realidade, o número da linha após **AT** não é completamente ignorado; tem que estar entre — 21 e + 21 ou um erro ocorrerá. Por essa razão é mais seguro especificar linha 0.

O item "**AT 21 — N,N**" na versão anterior do nosso programa seria muito melhor apesar de menos ilustrativa se substituído por "**AT 0, N**")

2. Faça uma impressão do gráfico do seno executando o programa do capítulo XVIII e usando **COPY**.

# CAPÍTULO XX

## Sub-strings

Dada uma *string*, uma *sub-string* consiste de um número de caracteres consecutivos da *string*, apanhados em sequência.

Há uma notação, chamada *SLICING*, para descrever *sub-strings* a qual pode ser aplicada a expressão *strings* arbitrárias. A forma geral é

### EXPRESSÕES STRING (INÍCIO TO FIM)

de tal forma que, por exemplo,

`"ABCDEF" (2 TO 5) = "BCDE"`

se você omitir o início, então é assumido 1; se omitir o fim, então o comprimento da *string* é assumido. Assim

`"ABCDEF" (TO 5) = "ABCDEF" (1 TO 5) = "ABCDEF"`

`"ABCDEF" (2 TO) = "ABCDEF" (2 TO 6) = "BCDEF"`

`"ABCDEF" ( TO ) = "ABCDEF" (1 TO 6) = "ABCDEF"`

(você pode também escrever essa última como `"ABCDEF" ( )`).

Uma pequena diferença de omitir o **TO** é ter apenas um número.

`"ABCDEF" (3) = "ABCDEF" (3 TO 3) = "C"`

Apesar de que normalmente ambos, o início e o fim, devam se referir a partes existentes da *string*, essa regra é superada por uma outra. Se o início é maior que o fim, então o resultado é um *astring* vazia. Assim

`"ABCDEF" (5 TO 7)`

Dá erro 3 (erro se subscripto) porque, como a *string* contém apenas 6 caracteres, 7 e demais.

`"ABCDEF" (8 TO 7) = " "`

e

`"ABCDEF" (1 TO 0) = " "`

O início e fim não podem ser negativos, ou você obtém erro B.

O próximo programa faz *B\$* igual a *A\$*, mas omite qualquer espaço em branco.

```
10 INPUT A$
20 FOR N = LEN A$ TO 1 STEP - 1
30 IF A$(N) <> " " THEN GOTO 50
40 NEXT N
50 LET B$ = A$ (TO N)
60 PRINT "*****"; A$ "*****"; B$;
*****
70 GOTO 10
```

Note que se *A\$* é inteiramente espaços, então na linha 50 nós temos `N = 0` e `A$ (TO N) = A$ (1 TO 0) = " "`

Para variáveis *string*, nós não só podemos extrair *sub-strings*, mas também atribuir *sub-strings*. Por exemplo, digite

`LET A$ = "ESTÁ MUITO FORA"`

e então `LET A$ (5 TO 8) = "*****"`  
e `PRINT A$`

Note que uma vez que a *sub-string* *A\$* (5 TO 8) é apenas 4 caracteres, os quatro primeiros asteriscos foram usados. Essa é uma característica de atribuição a *sub-strings*: a

sub-string, após a operação, tem que ter exatamente o mesmo tamanho que anteriormente. Para certificar-se de que isso aconteceu, a string que está sendo atribuída é cortada à direita se for muito grande ou preenchida com espaços, se for muito pequena. Isso é chamado de atribuição **PROCRUSTIANA**.

Se você agora tentar

```
LET A$ ( ) = "ESPLÊNDIDO"
PRINT A$, " "
```

você verá que a mesma coisa aconteceu novamente (dessa vez com espaço incluído) porque **A\$ ( )** conta como uma sub-string.

```
LET A$ = "ESPLÊNDIDO"
```

funcionará perfeitamente, então, o **SLICING** pode ser considerado como tendo prioridade 12; assim, por exemplo,

```
LEN "ABCDEF" (2 TO 5) = LEN "ABCDEF"
(2 TO 5) = 4
```

Expressões *string* complicadas necessitam de parênteses à sua volta, antes que uma operação de **SLICING** seja executada. Por exemplo:

```
"ABC" + "DEF" (1 TO 2) = "ABCDE"
("ABC" + "DEF") (1 TO 2) = "AS"
```

### Resumo

*Slicing*, usando **TO**; note que essa notação é exclusiva do NE-Z8000.

### Exercícios

1. Alguns tipos de BASIC (não o do NE-Z8000) tem 3 funções, chamadas **LEFT\$**, **RIGHT\$** e **MID\$**.

**LEFT\$ (A\$, N)** fornece a sub-string de **A\$** consistindo dos primeiros N caracteres.

**RIGHT\$ (A\$, N)** fornece a sub-string de **A\$** consistindo dos caracteres n em diante.

**MID\$ (A\$, N1, N2,)** fornece a sub-string de **A\$** consistindo de N2 caracteres iniciando no caractere N1.

Como você escreveria isso no Basic de 8k do NE-Z8000?

O Basic 4K do NE-Z80 tem uma função chamada **TL\$**. **TL\$ (A\$)** fornece uma sub-string de **A\$** formada de todos os caracteres, exceto o primeiro.

Como você escreveria isso no Basic 8k do NE-Z8000? Funcionaria como *strings* de tamanho 1?

2. Tente essa seqüência de comandos:

```
LET A$ = "X" + "Y"
LET A$ (2) = CHR$ 11 (o caractere aspas)
LET A$ (4) = CHR$ 11
PRINT A$
```

**A\$** é agora uma *string* com as aspas dentro dela! Tudo se passa como você tivesse digitado:

```
LET A$ = "X" + "Y"
```

A parte à direita do sinal de igualdade teria sido tratada como uma expressão, dando a **A\$** o valor "XY".

Agora digite

```
LET B$ = "X" + "Y"
```

Você verá que apesar de **A\$** e **B\$** parecem os mesmos quando impressos eles não são iguais — tente

```
PRINT A$ = B$
```

Onde **B\$** contém meias imagens de aspas (com código 192). **A\$** contém as genuínas aspas (com código 11).

3. Execute este programa:

```
10 LET A$ = "LEN" + " ABCD "
100 PRINT A$; " = "; VAL A$
```

Isso falhará, porque **VAL** não trata as imagens de aspas "" como aspas.

Insira algumas linhas entre a 10 e 100 para trocar as imagens de aspas em **A\$** por aspas (que você deve chamar **CHR\$ 11**) e tente novamente.

Faça o mesmo tipo de modificação para o programa no capítulo IX, exercício 3.

4. Digite a sub-rotina que ignora espaços, e escreva e execute um programa que a use.

5. Este programa cancela toda ocorrência da *string* "SUPERMAN" de **A\$**.

```
1000 FOR N = 1 TO LEN A$ - 7
1020 IF A$ (N TO N + 7) = "SUPERMEN"
THEN LET A$ (N TO N + 7) = "*****"
1030 NEXT N
1040 RETURN
```

Escreva um programa que dê a **A\$** vários valores (isto é, "SUPERMAN É FORTE") e aplique a sub-rotina.

# CAPÍTULO XXI

## Arrays

O array é um conjunto de variáveis, ou elementos, todos com o mesmo nome e distinguidos por um número (o subscrito) escrito entre parênteses após o nome. Por exemplo, o nome poderia ser *A* (como uma variável de controle de **LOOPS FOR-NEXT**, o nome de um array deve ser apenas uma letra) e doze variáveis seriam, então, *A*(1), *A*(2), e assim por diante até *A* (12).

Os elementos de um array são chamados **variáveis subscritas**, ao contrário das **variáveis simples**, com as quais você já está familiarizado.

Antes de você usar um array, você deve reservar espaço para ela no computador. Para fazer isso use a instrução **DIM** (de dimensão).

### **DIM A (12)**

Define um array chamado *A* com **dimensão** 12 [ou seja, há 12 valores subscritos *A*(1), . . . , *A*(12)], e gera os 12 valores. Também cancela qualquer array chamado *A* que existisse anteriormente (mas não uma variável simples. Um array e uma variável simples com os mesmos nomes podem coexistir. Não haveria confusões entre elas, porque uma variável array sempre tem um subscrito).

O subscrito pode ser uma expressão numérica arbitrária; agora você pode escrever

```
10 FOR N = 1 TO 12
20 PRINT A (N)
30 NEXT N
```

Você também pode definir *arrays* com mais de uma dimensão. Num *array* bidimensional você necessita dois números para especificar um elemento — como a linha e a coluna que especificam a posição do caractere na tela — e assim tem a forma de uma tabela. Alternativamente, se você imaginar os números da linha (**DUAS DIMENSÕES**) como referenciando a uma página impressa, você poderá usar uma dimensão extra para o número da página. Claro que estamos falando sobre *arrays* numéricos; os elementos não se-

riam caracteres impressos como em um livro, porém números. Pense no elemento de um *array* tridimensional *C* como sendo especificado por *C* (número da página, número da linha, número da coluna).

Por exemplo, para definir um *array* bidimensional *B* com dimensões 3 e 6, você usa a instrução **DIM**.

### **DIM B (3, 6)**

Isto então fornece  $3 \times 6 = 18$  variáveis subscritas

*B* (1,1), *B* (1,2), . . . , *B* (1,6)

*B* (2,1), *B* (2,2), . . . , *B* (2,6)

*B* (3,1), *B* (3,2), . . . , *B* (3,6)

O mesmo princípio funciona para qualquer número de dimensões.

Apesar de você poder ter um número e um *array* com o mesmo nome, você não pode ter dois *arrays* com o mesmo nome, mesmo se eles tiverem números diferentes de dimensões.

Há também *arrays* tipo *string*. As *strings* em um *array* diferem de *strings* simples por serem de tamanho fixo e a atribuição a elas é sempre **PROCRUSTIANA**. Uma outra maneira de considerá-las é como *arrays* (com uma dimensão extra) de um único caractere. O nome de um *array string* é apenas uma letra seguida de *S*, e um *array string* e uma *string simples* não podem ter o mesmo nome (ao contrário do caso dos números).

Suponha, então, que você quer um *array AS* de cinco *strings*. Você deve decidir qual o tamanho que essas *strings* devem ter — suponhamos que 10 caracteres para cada uma seja suficiente.

Então,

### **DIM AS (5,10)** (digite isso)

Isso define um *array* 5 x 10 de caracteres, mas pode também considerar cada fileira como sendo um *string*.

*AS* (1) = *AS* (1,1) *AS* (1,2) . . . *AS* (1,10)

*AS* (2) = *AS* (2,1) *AS* (2,2) . . . *AS* (2,10)

*AS* (5) = *AS* (5,1) *AS* (5,2) . . . *AS* (5,10)

Se você fornecer o mesmo número de subscritos (dois no caso) que foram dimensionados na instrução **DIM** você obtém apenas um caractere; porém se você omitir o último, você obtém a *string* de tamanho fixo. Assim, por exemplo, **AS (2,7)** é o sétimo caractere na *string* **AS (2)**.

Usando uma noção de *slicing*, nós poderíamos também escrever isso como **AS (2) (7)**.

Agora digite.

```
LET AS (2) = "1234567890"
```

e

```
PRINT AS(2), AS (2,7)
```

você obtém

```
1234567890      7
```

Para o último subscrito (o que pode ser omitido), você também pode ter uma operação de *slicing*, de maneira que, por exemplo.

```
AS (2,4 TO 8) =
```

```
AS (2) (4 TO 8) = AS (2) (4TO 8) = "45678"
```

Lembre-se

Numa *string array*, todas as *strings* têm o mesmo tamanho fixo.

A instrução **DIM** tem um número extra (o último) para especificar seu comprimento.

Quando você escreve uma variável subscrita para um *array*, você pode acrescentar um número, ou uma operação *slicing*, para corresponder ao número extra na instrução **DIM**.

**Resumo**

*Arrays*

Instrução **DIM**

**Exercícios**

1. Defina um *array MS* de doze *strings*, na qual *MS* é o nome do primeiro mês.

(DICA: a instrução **DIM** será **DIM MS (12,9)**. Teste imprimindo todos os *MS* (F) (Use um **LOOP**). Digite:

```
PRINT "este é o mês de"; MS (5);
```

```
"O mês de Maria".
```

O que pode ser feito sobre aqueles espaços?

2. Você pode ter um *array string* sem dimensão. Digite:

```
DIM AS (10)
```

e você verá que **AS** comporta-se como uma variável *string*, exceto por ter sempre tamanho 10 e a atribuição será sempre

**PROCRUSTIANA.**

### 3. READ, DATA E RESTORE

A maioria dos Basics (mas não o Basic 8 k do NE-Z8000) tem três instruções chamadas **READ**, **DATA** e **RESTORE**.

Uma instrução **DATA** é uma lista de expressões. Toda instrução **DATA** no programa fornece lista de expressões, uma por uma.

**READ X**

Por exemplo, atribui o valor da expressão corrente na **DATA LIST** à variável *x* e move para a próxima expressão, para a próxima instrução **READ**.

**RESTORE** (Volta ao início da **DATA LIST**)

Em teoria, você pode trocar as informações **READ** e **DATA** pela instrução **LET**; entretanto, um de seus maiores usos é para inicialização de *arrays*, como neste programa:

```
5 REM ESTE PROGRAMA NÃO FUNCIONA  
NO BASIC 8K DO NE-Z8000
```

```
10 DIM MS (12,3)
```

```
20 FOR N = 1 TO 12
```

```
30 READ MS (N)
```

```
40 NEXT N
```

```
50 DATA "JAN", "FEV", "MAR", "ABR"
```

```
60 DATA "MAI", "JUN", "JUL", "AGO"
```

```
70 DATA "SET", "OUT", "NOV", "DEZ"
```

Se você desejar executar este programa apenas uma vez, você deve substituir a linha 30 por

```
30 INPUT MS (N)
```

e você não terá nenhuma outra digitação a fazer. Entretanto, se você deseja preservar o programa, você certamente não desejaria digitar os meses cada vez que for executá-lo.

Nós sugerimos este método:

(I) Inicialize o *array*, usando um **LOOP FOR-NEXT** e uma instrução de **INPUT** como foi descrito acima.

(II) Cancele o **LOOP FOR-NEXT** e a instrução **INPUT** (porém não com **NEW**, porque você deseja preservar o *array*).

(III) Digite o resto do programa e preserve-o. Isso salvará as variáveis, incluindo o *array*.

(IV) Quando você recarregar o programa, você também carregará o *array*.

(V) Quando executar o programa, não use **RUN**, o qual limpa as variáveis. Use **GOTO**.

## CAPÍTULO XXII

### Quando o computador fica repleto

O NE-Z8000 tem um armazenamento integral limitado, e isso não é difícil de notar. A melhor indicação que a área de um armazenamento está cheia é normalmente uma denotação de erro 4. Porém, outras coisas podem acontecer, e algumas delas são muito estranhas. Se você tem uma expansão de memória (RAM) desacople-a por um momento.

O **Arquivo de imagem**, ou seja, a área onde o computador guarda as imagens da televisão, é engenhosamente desenhado, de tal maneira a só ocupar espaço com o que foi impresso até então. Uma linha de imagem é formada de até 32 caracteres e um *character enter*. Isso significa que você pode ficar sem espaço imprimindo algo, e o momento mais óbvio é quando faz uma listagem. Digite:

```
NEW  
DIM A (150)  
10 FOR I = 1 TO 15  
20 PRINT I
```

Aqui aparece a primeira surpresa: a linha 10 desaparece da listagem. A listagem deveria incluir a linha corrente, 20, e não há espaço na memória para as duas linhas. Agora digite:

```
30 NEXT I
```

Normalmente, só há espaço para a linha 30 na listagem. Agora digite:

```
40 REM X (sem ENTER)
```

E você verá a linha 30 desaparecer e a linha 40 saltar para o topo da tela. Você ainda tem o cursor **█** e pode movê-lo. Tudo que você viu é algum mecanismo obscuro que dá à metade inferior da tela 24 linhas para dar-lhe prioridade sobre a metade superior. Agora digite:

```
XXXXXX (ainda sem ENTER)
```

E o cursor desaparecerá — não há espaço para imprimi-lo. Digite outro X, sem ENTER.

Você verá que o último caractere X irá desaparecer. Agora digite ENTER e tudo desaparecerá; o programa, porém, ainda se encontra no computador, como você pode comprovar cancelando a linha 10 e usando as teclas **←** e **→** para verificar as linhas do mesmo.

```
10 FOR I = 1 TO 15
```

Novamente essa linha irá se mover para o topo da tela, como fez a linha 40. Porém quando você pressionar ENTER ela não entrará, apesar de não ter mensagens de erro ou **S** indicador de erro de sintaxe, para indicar mais algum erro. É o resultado de não haver espaço na memória para verificar a sintaxe de linhas que contém números (outro além do número da linha no início).

A solução é fazer espaço, de alguma maneira, mas, primeiramente, cancele a linha 10 e pressione EDIT. A tela ficará branca, porque não há espaço para trazer o programa para a tela.

Pressione ENTER e você obterá parte da listagem. Agora cancele a linha 40 (a qual você não queria, de qualquer maneira) digitando:

#### 40 (e ENTER)

Agora digitando a linha 10 novamente — ela ainda não será aceita. Cancele-a novamente; você de alguma maneira ainda tem que conseguir espaço.

Tenha em mente que a razão pela qual a linha 10 foi rejeitada foi provavelmente porque não havia espaço para verificar a sintaxe dos dois números, 1 e 15. Então cancele a linha 20 do programa; você deve ter espaço para entrar com a linha 10 e para entrar com a linha 20 (que não contém números).

Tente isto; digite:

```
20
10 FOR N = 1 TO 20
20 PRINT n
```

e o programa está digitado corretamente. Digite:

#### GOTO 10

novamente você verá que esta linha não é aceita, pois sua sintaxe não pode ser verificada. Entretanto, se você cancelar e digitar:

#### RUN

funcionará (RUN limpa o *array*, abrindo bastante espaço).

Agora digite o mesmo que antes, desde o NEW até a linha 30, e então:

```
40 REM xxxxxxxxxxxxxx
```

(12 vezes x) o qual acabará parecendo com 40 RE. Quando você pressionar ENTER, a listagem será formada apenas pela linha 30, e na realidade a linha 40 terá sido totalmente perdida. Isso porque ela foi simplesmente muito longa para caber no programa. O resultado é um pouco pior quando uma linha é uma nova versão de uma linha já existente no programa; você perderá ambas: a linha antiga do

programa e a nova, que devia substituí-la.

A solução para esse problema é adquirir uma placa de expansão de memória RAM, a qual encaixa na parte traseira de seu computador.

A expansão de 16k NE-X dá ao computador 16 vezes mais memória do que ele tem na sua forma normal.

O comportamento com a placa de expansão é bastante diferente, porque o arquivo de imagem é preenchido com espaços para fazer com que cada linha tenha 32 caracteres.

Agora imprimir e listar não fará com que seu computador fique sem memória, e você não verá essas listagens simplificadas e saltos de linhas, mas verá linhas se perdendo e novamente a solução será conseguir espaço:

Se você tem uma expansão de memória, acople-a e siga a digitação desse capítulo, usando

#### DIM A (3069)

para substituir DIM A (150).

Para resumir,

1. se a listagem começa a ficar resumida ou as coisas começam a saltar pela tela, você está ficando sem espaço.
2. se ENTER parece não ter efeito no fim de uma linha, provavelmente não há espaço para lidar com um número. Cancele a linha, usando EDIT — ENTER ou DELETE
3. ENTER pode por a perder toda uma linha.

Para todas essas coisas a solução é a mesma: não se apavore, e procure espaço livre.

A primeira coisa a fazer é considerar o CLEAR. Se você tem variáveis e você não se importa de perdê-las, esta é a coisa a fazer.

Isso falhando, procure instruções desnecessárias no programa, tais como REM e cancele-as.

#### Resumo

Quando a memória lotar, coisas terríveis podem acontecer; mas normalmente elas não são fatais.

# CAPÍTULO XXIII

## Contando

Apesar dos engenheiros usarem um sistema binário quando estão construindo computadores (veja a coluna da direita na tabela a seguir), um outro sistema numérico, o Hexadecimal, tendo base 16, é útil porque é mais fácil de ler e pode facilmente ser convertido para binário. Ele começa assim:

### HEXADECIMAL PORTUGUÊS

0	zero
1	um
2	dois
.	.
.	.
.	.
9	nove
A	dez
B	onze
C	doze
D	treze
E	Quatorze
F	Quinze
10	Dezesseis
11	Dezessete
.	.
.	.
.	.
19	Vinte e Cinco
1A	Vinte e Seis
1B	Vinte e Sete
.	.
.	.
.	.
1F	Trinta e um
20	Trinta e Dois
21	Trinta e Três
.	.
.	.
.	.

9E	Cento e Cinquenta e Oito
9F	Cento e Cinquenta e Nove
40	Cento e Sessenta
41	Cento e Sessenta e um
.	.
.	.
.	.
FE	Duzentos e cinquenta e quatro
FF	Duzentos e Cinquenta e Cinco
100	Duzentos e Cinquenta e Seis

Se você estiver usando notação **HEXADECIMAL** e quer deixar o fato bem claro, escreva "h" no final dos números. Por exemplo, para cento e cinquenta e oito, escreva "9Eh" e leia "nove E hexa".

Em vários sistemas numéricos, a contagem começa assim:

PORTUGUÊS	DECIMAL	HEXADECIMAL	BINÁRIO
ZERO	0	0	0 ou 0000
UM	1	1	1 ou 0001
DOIS	2	2	10 ou 0010
TRÊS	3	3	11 ou 0011
QUATRO	4	4	100 ou 0100
CINCO	5	5	101 ou 0101
SEIS	6	6	110 ou 0110
SETE	7	7	111 ou 0111
OITO	8	8	1000
NOVE	9	9	1001
DEZ	10	A	1010
ONZE	11	B	1011
DOZE	12	C	1100
TREZE	13	D	1101
QUATORZE	14	E	1110
QUINZE	15	F	1111
DEZESSEIS	16	10	10000

O ponto importante é que dezesseis é igual a 2 elevado à quarta potência, o que torna conversão entre HEXADECIMAL e BINÁRIO muito fácil.

Para converter de HEXADECIMAL para BINÁRIO, troque cada dígito HEXADECIMAL por quatro BITS, usando a tabela acima. Os dígitos binários 0 e 1 são considerados como BITS.

Para converter binário em hexadecimal, divida o número binário em grupos de quatro, iniciando pela direita, e então troque cada grupo por um correspondente em hexadecimal.

Os BITS dentro do computador estão na maioria agrupados em conjuntos de oito, ou *bytes*. Um único *byte* pode representar qualquer número de zero a 255 (11111111 ou FF HEXA); ou, qualquer caractere do conjunto de caracteres do NE-Z8000. Seu valor pode ser escrito com dois dígitos hexadecimais.

Dois *bytes* podem ser agrupados para formar o que é

tecnicamente chamado de **palavra**. Uma palavra pode ser escrita usando 16 *bits* ou quatro dígitos hexadecimais e representa um número de 0 até (em decimal)  $2^{16}-1 = 65535$ .

Um *byte* tem sempre 8 *bits*, mas palavras variam de computador para computador.

#### Resumo

Sistemas DECIMAL, HEXADECIMAL e BINÁRIO, BITS e BYTES. (não os confunda) e palavras.

#### Exercícios

1. Como você faria conversão entre DECIMAL e HEXADECIMAL?

Escreva um programa no NE-Z8000 para converter valores numéricos em strings, dando sua representação em HEXADECIMAL e vice-versa (isto é o que **STR\$** e **VAL** fazem da representação decimal.).

## CAPÍTULO XXIV

### Como funciona o computador

Não é objetivo deste manual descrever em detalhes a eletrônica do NE-Z8000 e sua operação, mas podemos dar uma idéia da função de seus componentes.

A ilustração mostra o NE-Z8000 com sua caixa removida (nota: em caso da remoção da caixa, haverá perda da garantia). Os componentes retangulares com pernas de metal são CIs — circuitos integrados; na realidade, você está olhando apenas o encapsulamento — o circuito é muito menor.

O CI mais importante do NE-Z8000 é a CPU (Unidade Central de Processamento), que é um microprocessador Z80A. O processador faz a aritmética, e eletronicamente controla o resto do computador de acordo com o programa do sistema operacional.

O sistema operacional está contido numa memória EPROM, isto é, num dispositivo eletrônico de armazenamento que tem um programa permanente para fazer a CPU funcionar.

O programa em forma simbólica é uma longa sequência de BYTES (BYTE é um número entre 0 e 255). Cada Byte tem um ENDEREÇO mostrando sua posição na EPROM. O primeiro tem endereço 0, o segundo tem endereço 1, e assim por diante, até 8191, porque o NE-Z8000 tem um Basic de 8k.

Você pode ver que BYTE está em um determinado endereço, usando a função PEEK. Por exemplo, este programa imprime os primeiros 21 BYTES da EPROM (e seus endereços).

```
10 PRINT "ENDEREÇO"; TAB 8; "BYTE"  
20 FOR A = 0 TO 20  
30 PRINT A; TAB 8; PEEK A  
40 NEXT A
```

A memória RAM é um "BLOCO DE RASCUNHO" eletrônico que está ligado à CPU. O programa Basic que você digita é armazenado eletronicamente lá, como são as va-

riáveis do programa, a imagem da televisão, e as variáveis do sistema.

Como na EPROM, o armazenamento na RAM é feito em BYTES, cada um com um endereço; esses variam de 16384 a 17407 (ou 32767, se você tiver uma extensão de 16k). Como na EPROM, você pode encontrar os valores desses BYTES usando PEEK. A diferença é que você também pode mudá-los. Digite:

```
POKE 17300, 57
```

Isso dá ao Byte no endereço 17300 o valor 57. Se você agora digitar

```
PRINT PEEK 17300
```

você obterá o número 57 novamente (tente fazer POKE de outros valores). Note que o endereço tem que estar entre 0 e 65535 e a maioria desses se referirá a BYTES na EPROM. O valor deve estar entre -255 e +255 e, se for negativo, soma 256.

A possibilidade do POKE lhe dá um poder imenso ao lidar com o computador, se você souber como usá-lo; entretanto, o conhecimento necessário é muito maior do que pode haver num manual introdutório como este.

Afora os componentes acima descritos, existem outros circuitos integrados de menor complexidade, que desempenham várias funções lógicas. Esses circuitos integrados são essenciais para o funcionamento do computador.

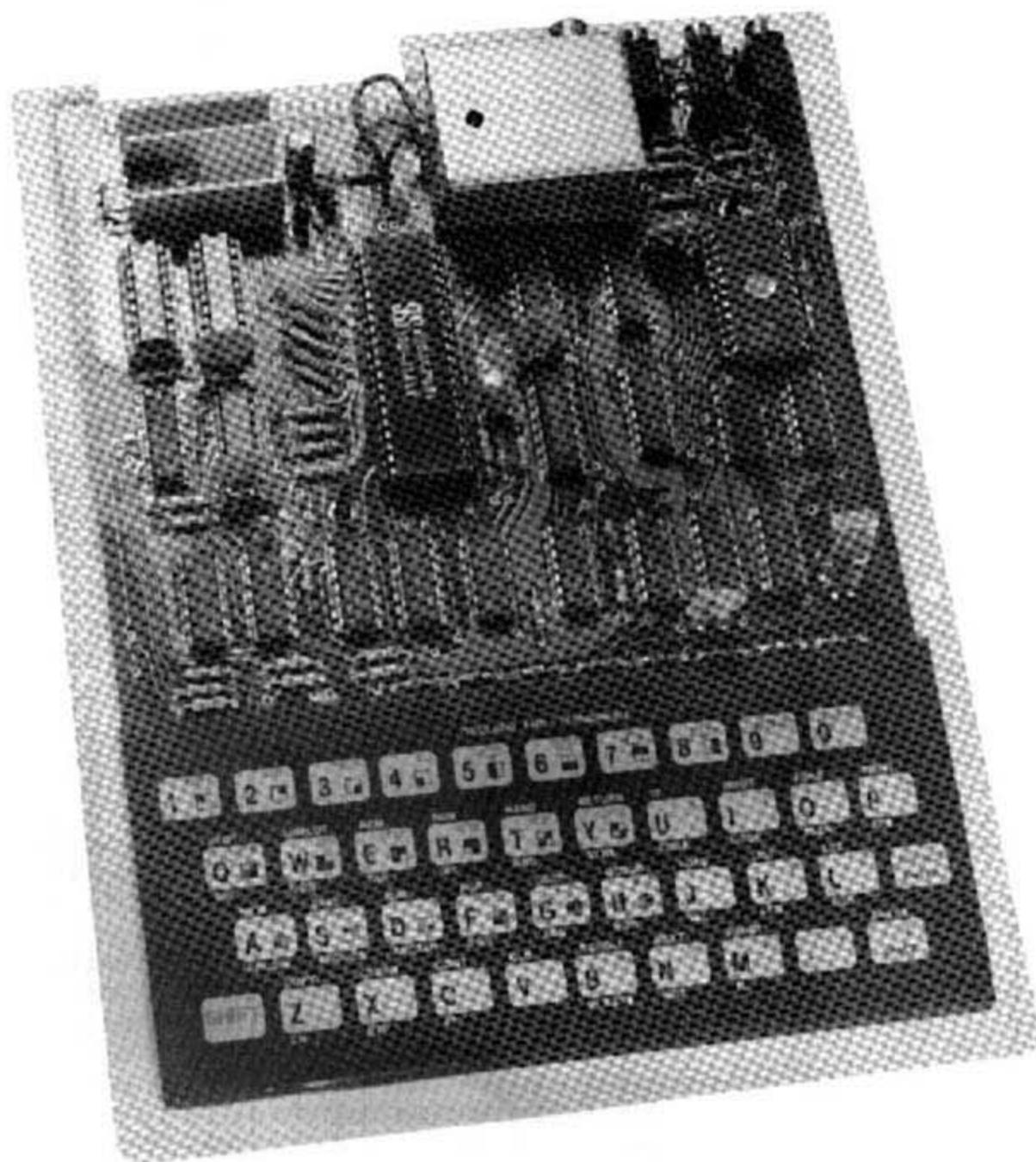
O modulador converte a saída do computador para a televisão em uma forma adequada para a televisão, e o regulador converte os 9 volts contínuos em 5 volts regulados.

#### Resumo

#### CIRCUITO

Instruções: POKE

Funções: PEEK



## CAPÍTULO XXV

# Empregando linguagem de máquina

Este capítulo é escrito para aqueles que entendem a linguagem de máquina do Z80, o conjunto de instruções que o Z80 usa.

Se você deseja aprendê-la, o melhor meio é consultar o **Z80 ASSEMBLY LANGUAGE PROGRAMMING MANUAL**, juntamente com o **Z-80-CPU, Z-80 A-CPU TECHNICAL MANUAL**, publicado pela ZILOG; mas esses não são recomendados para principiantes.

Você pode procurar, também, livros sobre o Z-80. Existem alguns em inglês ou outros idiomas. Existe pouca coisa em português sobre o assunto.

Rotinas em linguagem de máquina podem ser executadas em um programa BASIC usando a função **USR**. O argumento de **USR** é o endereço de início da sub-rotina, e seu resultado é um inteiro de dois Bytes, sem sinal, o do par de registradores **bc** em retorno. O endereço de retorno para o BASIC é mantido da forma normal; assim o retorno é feito através de uma instrução **RET** do NE-Z8000.

Há algumas restrições em rotinas **USR**:

(F) Em retorno, os registradores **iy** e **i** devem ter valor  $4000h$  e  $1Eh$ .

(II) A rotina de *display* usa os registradores **a'**, **f'**, **ix**, **iy**; uma rotina **USR** não os deve usar se tiver operando na modalidade **slow** (ver apêndice C) (também não é seguro ler o par **af'**).

Todas as linhas do processador estão expostas na parte traseira do NE-Z8000. Assim, a princípio você pode fazer tudo que fez com o Z-80 no NE-Z8000 aqui está um diagrama das partes expostas:



Uma parte do código de máquina no meio da memória corre o risco de ser coberto pelo sistema BASIC.

Alguns pontos mais seguros são:

(I) Em uma instrução **REM**: Digite uma instrução **REM** com caracteres bastante para conter os códigos de máquinas, o qual você então fará um **POKE**. Evite instruções de **HALT**, uma vez que isto será reconhecido como fim da instrução **REM**.

(II) Em uma *string*: Defina uma longa *string* e então atribua um código de máquina a cada caractere.

Em ambos os casos, o código está salvo, mas sujeito a se mover; este é especialmente o caso com *strings*. No apêndice A, o conjunto de caracteres, você encontrará os caracteres e instruções do Z-80 escritas um ao lado do outro, e você os achará úteis quando tiver que entrar com os códigos.

(III) — No topo da memória: NE-Z8000 é ligado, ele faz testes para ver quanta memória há e coloca a pilha da máquina exatamente no topo, de forma que não há espaço para rotinas. **USR** — Lá ele guarda o endereço do primeiro Byte não existente (isto é,  $17k$  ou  $17408$ , se você tiver  $1k$  de memória), em uma variável do sistema conhecida como **RAMTOP**, nos dois Bytes com endereços  $16388$  e  $16389$ . **NEW** por outro lado, não faz um teste de memória cheia, mas apenas verifica até antes do endereço em **RAMTOP**. Assim, se você fizer um **POKE** do endereço de um byte existente para **RAMTOP**, para **NEW** toda memória daquele *byte* em diante está fora do sistema BASIC e é deixada de lado. Por exemplo, suponha que você tem  $1k$  de memória e você acabou de ligar o computador.

```
PRINT PEEK 16388 + 256 * PEEK 16389
```

fornece o endereço ( $17408$ ) do primeiro **BYTE** inexistente.

Agora, suponha que você tem uma rotina **USR** de  $20$  bytes, você quer mudar **RAMTOP** para  $17388 = 236 + 256 * 67$  (como você faria isso no computador? Assim, digite:

**POKE 16388, 236**  
**POKE 16389, 67**

e então **NEW**. Os vinte Bytes de memória, do endereço 17388 até 17407, agora são seus para você fazer o que quiser. Se digitar **NEW** novamente, não afetará esses vinte Bytes.

O topo da memória é um bom local para rotinas **USR**; seguro (até mesmo do **NEW**) e imóvel. A principal desvantagem é que não é preservado pelo **SAVE**.

#### **Resumo**

Funções: **USR**

Instruções: **NEW**

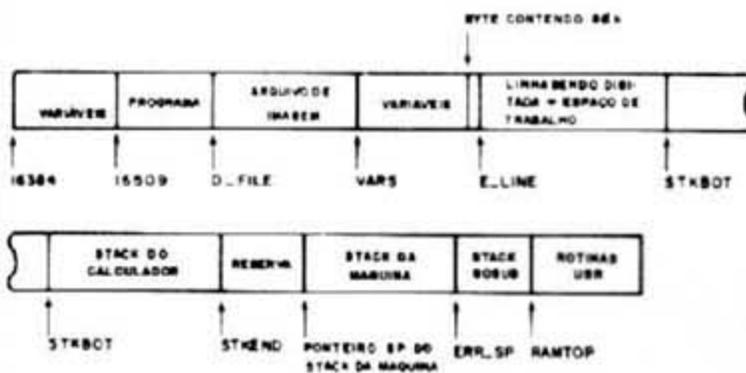
#### **Exercícios**

**1. Faça RAMTOP** igual a 16700 e então execute **NEW**. Você terá uma idéia do que acontece quando a memória lota.

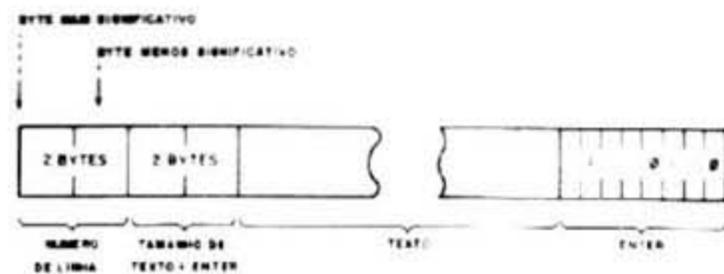
# CAPÍTULO XXVI

## Organização da armazenagem

A memória é dividida em diferentes áreas para armazenar tipos diferentes de informações. As áreas são de tamanho suficiente para conter apenas as informações que elas realmente contêm; e se você inserir mais, em um determinado local (por exemplo, adicionando uma linha de programa ou variável), o espaço é criado deslocando-se tudo acima desse ponto. Da mesma forma, se você cancelar a informação, tudo acima do cancelamento é deslocado para baixo.



As variáveis do sistema contêm muitas informações que dizem ao computador o estado no qual ele se encontra. Elas estão todas listadas no próximo capítulo, mas, por enquanto note que há algumas (chamadas **D\_FILE**, **VARS**, **E\_LINE**, por exemplo) que contêm os endereços de limite entre as várias áreas da memória. Essas não são variáveis, e seus nomes não serão reconhecidos pelo computador. No programa, cada linha é armazenada como



Note que, em oposição a todos os outros casos de número de dois bytes, no **Z 8H**, o número da linha aqui (e também numa variável de controle de **LOOP FOR-NEXT**) é armazenada com seu primeiro byte, mais significativo, isto é, na ordem na qual você os escreveria.

Uma constante numérica no programa é seguida por sua forma binária, usando o caractere **CHR\$126**, seguido de **5 BYTES** para o número.

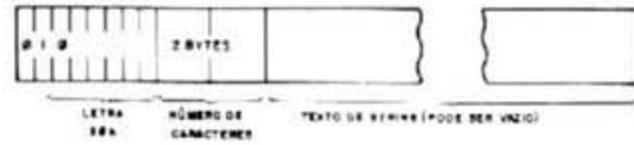
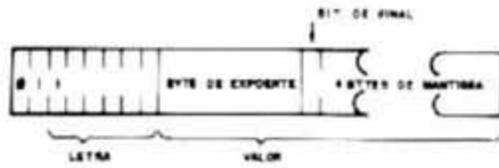
O arquivo de imagem é a cópia, na memória, da imagem da televisão. Começa com um caractere **ENTER**, seguido de 24 linhas de texto, cada uma terminando com **ENTER**. O sistema é desenhado de forma que a linha de texto não necessite sempre de 32 caracteres. Os espaços finais podem ser omitidos. Isso é feito para economizar espaço quando a memória é pequena.

Quando o total de memória (de acordo com a variável de sistema **RAMTOP**) é menor que  $3\frac{1}{4}$  K, uma limpeza de tela — como é feito no início ou por um **CIS** — consiste de apenas 25 **ENTERS**. Quando a memória é maior uma limpeza de tela é feita com  $24 \times 32$  espaços; o **SCROLL**, entretanto, e certas condições onde a parte inferior da tela se expande para mais que duas linhas, podem perturbar isto, produzindo linhas curtas na parte inferior da tela.

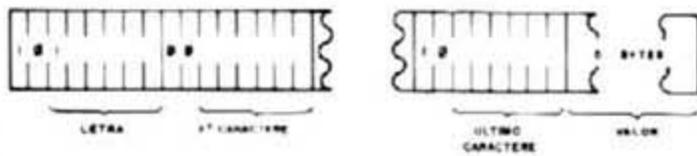
As variáveis têm formato diferentes, dependendo de sua natureza.

O nome do número, se for apenas uma letra, será

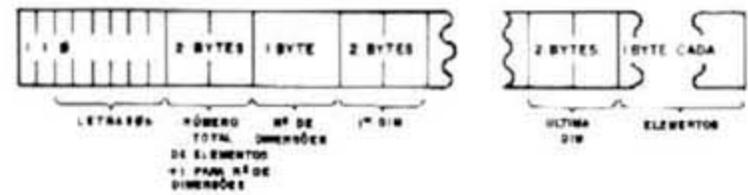
STRING:



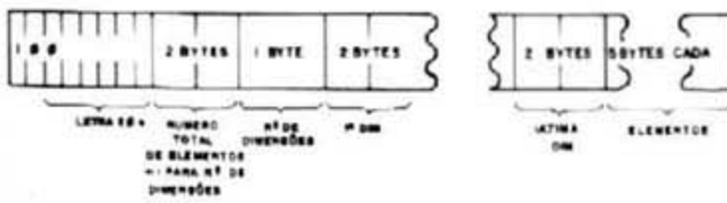
Se o nome de um número for mais que uma letra



ARRAY DE CARACTERES:



ARRAY DE NÚMEROS



A ordem dos elementos é:

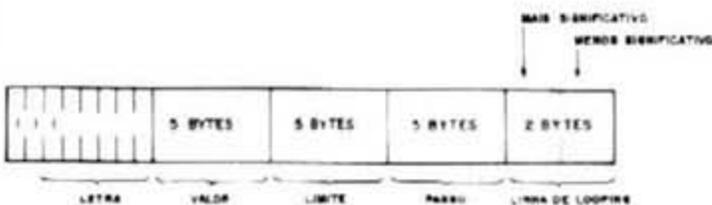
Primeiro, o elemento para o qual o primeiro subscrito é 1. Em seguida, o elemento para o qual o primeiro subscrito é 2. Em seguida, os elementos para os quais o primeiro subscrito é 3

E assim por diante para todos os valores possíveis do primeiro subscrito.

Os elementos com o primeiro subscrito dado estão ordenados da mesma maneira usando o segundo subscrito, e assim por diante, até o último.

Como exemplo, o elemento da *array* B 3x6, no capítulo 21, estão armazenados na ordem B (1,1), B(1,2) B(1,3) B(1,4) B(1,5), B(1,6), B(2,1), B(2,2), . . . , B(2,6), B(3,1), B(3,2), . . . , B(3,6).

Variável de controle de um LOOP FOR-NEXT:



A parte iniciando em E\_LINE contém a linha sendo digitada (como um comando, uma linha de programa, um dado de entrada) e também algum espaço de trabalho.

A calculadora é parte do sistema BASIC que lida com aritmética, e os números que estão em operação são tratados em sua maioria no STACK calculador. A parte livre contém o espaço até então não usado. O STACK de máquina é o STACK usado pelo CI Z-80 para manter o endereço de retorno e assim por diante.

O espaço para rotina USR teve que ser separado por você, usando NEW como foi descrito no último capítulo.

## CAPÍTULO XXVII

### Variáveis do sistema

Os bytes na memória, de 16384 a 16508, são deixados para uso específico do sistema. Você pode fazer um **PEEK** deles para descobrir várias coisas sobre o sistema, e algumas delas podem ser úteis. Elas se encontram listadas aqui com seus usos.

Essas são as chamadas variáveis de sistema, mas não as confunda com as variáveis usadas pelo BASIC. Você não pode usar os mesmos nomes em um programa BASIC; são apenas mtemônicos que são usados para tornar mais fácil o referenciamento das variáveis.

As abreviações na coluna 1 tem os seguintes significados:

X não se pode fazer um **POKE**, porque será destruído pelo sistema.

N fazer um **POKE**, a variável não terá efeito  
S as variáveis são preservadas pelo **SAVE**.

O número da coluna 1 é o número de Bytes da variável. Para 2 Bytes, o primeiro é o menos significativo — o contrário do que você esperaria. Para fazer um **POKE** de um valor N para uma variável no endereço n, use, então:

**POKE** n,v — 256\* INT (v/256)

**POKE** n + 1, INT (v/256)

E para fazer um **PEEK** de seu valor, use a expressão

**PEEK** n + 256\* **PEEK** (n + 1)

NOTAS	ENDEREÇO	NOME	CONTEÚDO
1	16384	<b>ERR_NR</b>	1 a menos que o código de denotação. Inicia em 255 (para—1), assim <b>PEEK</b> 16384, se funcionar, dará 255. <b>POKE</b> 16384, n pode ser usado para forçar um erro <b>HALT</b> : #≤n≤14 dá uma das denotações, 15≤n≤34 ou 99≤n≤127 dá uma denotação nas standard e 35≤n≤98 provavelmente bagunçará o arquivo de imagem.
XI	16385	<b>FLAGS</b>	Vários Flags para controlar o sistema BASIC
X2	16386	<b>ERR_SP</b>	Endereço do primeiro item no Stack de máquina (após <b>GOSUB</b> retornar).

2	16388	<b>RAMTOP</b>	Endereço do primeiro Byte acima da área de sistema Basic. Você pode fazer um <b>POKE</b> a fim de que o <b>NEW</b> reserve espaço acima daquela área (veja capítulo 25) ou para enganar <b>CIS</b> para reservar um arquivo de imagem menor (capítulo 26)
N1 N2	16390 16391	<b>MODE</b> <b>PPC</b>	Especifica cursor <b>R</b> , <b>L</b> , <b>F</b> ou <b>G</b> Número da linha da instrução que está sendo executada. Um <b>POKE</b> não tem consequências, exceto na última linha do programa.
S1	16393	<b>VERSN</b>	Ø identifica EPROM de 8K em programas salvos.
S2	16394	<b>E_PPC</b>	Número da linha atual.
SX2	16396	<b>D_FILE</b>	Veja capítulo 26.
S2	16398	<b>DF_CC</b>	Endereço da posição do <b>PRINT</b> no arquivo de imagem. Pode ser feito um <b>POKE</b> de tal maneira que uma saída <b>PRINT</b> seja mandada para outro lugar.
SX2	16400	<b>VAR</b>	Veja capítulo 26
SN2 SX2	16402 16404	<b>DEST</b> <b>E_LINE</b>	Endereço da variável em atribuição. Veja capítulo 26
SX2	16406	<b>CH_ADD</b>	Endereço do próximo caractere a ser interpretado; o caractere após o argumento do <b>PEEK</b> , ou o <b>ENTER</b> no fim de uma instrução <b>POKE</b> .
S2	16408	<b>X_PTR</b>	Endereço do caractere precedendo o indicador <b>S</b>
SX2	16410	<b>STKBOT</b>	Veja capítulo 26
SX2	16412	<b>STKEND</b>	Veja capítulo 26.
SN1	16414	<b>BERG</b>	Registrador b do calculador
SN2	16415	<b>MEM</b>	Endereço usado para cálculos na memória (normalmente MEMBOT, mas nem sempre).
S1	16417	<b>NÃO USADO</b>	
SX1	16418	<b>DF_SZ</b>	Número de linhas (incluindo uma linha em branco) na parte inferior da tela.
S2	16419	<b>S_TOP</b>	O número da linha do topo do programa em listagem automática.
SN2 SN1	16421 16423	<b>LAST_K</b>	Mostra quais teclas foram pressionadas. Estado de <i>debounce</i> do teclado.
SN1	16424	<b>MARGIN</b>	Número de linhas em branco acima ou abaixo da imagem — 31.
SX2	16425	<b>NXTLIN</b>	Endereço da próxima linha de programa a ser executada.

S2	16427	<b>OLDPPC</b>	A linha para a qual <b>CONT</b> salta.
SN1	16429	<b>FLAGX</b>	Flags variados.
SN2	16430	<b>STRLEN</b>	Tamanho da designação do tipo de <i>string</i> em atribuição.
SN2	16432	<b>T_ADDR</b>	Endereço do próximo item na tabela de sintaxe (improvável que seja útil).
S2	16434	<b>SEED</b>	Dá origem ao <b>RND</b> . Esta é a variável atualizada pelo <b>RAND</b> .
S2	16436	<b>FRAMES</b>	Conta os quadros apresentados na televisão. BIT 15 é 1. BITS de 0 a 14 são decrementados para cada quadro enviado à televisão. Isso pode ser usado para temporização, mas <b>PAUSE</b> também a usa. <b>PAUSE</b> reajusta o bit 15 para 0 e coloca nos bits 0 a 14 o tamanho da pausa. Então quando tiverem chegado a zero, a pausa termina. Se a pausa ( <b>PAUSE</b> ) parar devido a uma tecla ter sido pressionada, o bit 15 é ajustado para 1 novamente.
S1	16438	<b>COORDS</b>	Coordenada X do último ponto plotado.
S1	16439		Coordenada Y do último ponto plotado.
S1	16440	<b>PR_CC</b>	O BYTE menos significativo do endereço da próxima posição para <b>LPRINT</b> imprimir ( <b>PRDUF1</b> ).
SX1	16441	<b>S_POSN</b>	Número da coluna para posição do <b>PRINT</b> .
SX1	16442		Número de linha para posição do <b>PRINT</b> .
S1	16443	<b>CDFLAG</b>	FLAGS variados. O bit 7 está ativo (1) durante a modalidade <b>SLOW</b> .
S33	16444	<b>PRBUFF</b>	<i>BUFFER</i> da impressora (33º caracteres é <b>ENTER</b> )
SN30	16477	<b>MEMBOT</b>	Area de memória para cálculo, usada para armazenar os números que não podem ser colocados convenientemente no <b>STACK</b> do calculador.
S2	16507	<b>NÃO USADO</b>	

#### Exercícios

1. Tente este programa

```

10 FOR N = 0 TO 21
20 PRINT PEEK (PEEK 16400 + 256 * PEEK
16401 + N)
30 NEXT N

```

Isso mostra os primeiros 22 bytes da área de variáveis. Tente combinar as variáveis de controle N com a discussão no capítulo 26.

2. No programa acima, troque a linha 20 para  
20 PRINT PEEK (16509 + N)

Isso mostra os 22 primeiros bytes da área de programa. Combine isso com o próprio programa.

# APÊNDICE A

## O conjunto de caracteres

Este é o conjunto completo de caracteres do NE-Z8000, com códigos em decimal e hexadecimal. Imaginando os códigos sob a forma de linguagem de máquina do Z80, vamos encontrar os mnemônicos correspondentes na coluna à direita. Como você deve saber, algumas instruções do Z80 são compostas, começando com CBh ou EDh, como se pode verificar nas duas colunas da direita.

<i>Código</i>	<i>Caracter</i>	<i>Hexa- decimal</i>	<i>Z80 assembler</i>	<i>após CB</i>	<i>após ED</i>
0	espaço	00	nop	rlc b	
1	■	01	ld bc,NN	rlc c	
2	■	02	ld (bc),a	rlc d	
3	■	03	inc bc	rlc e	
4	■	04	inc b	rlc h	
5	■	05	dec b	rlc l	
6	■	06	ld b,N	rlc (hl)	
7	■	07	rlea	rlc a	
8	■	08	ex af,af'	rre b	
9	■	09	add hl,bc	rre c	
10	■	0A	ld a,(bc)	rre d	
11	■	0B	dec bc	rre e	
12	£	0C	inc c	rre h	
13	\$	0D	dec c	rre l	
14	:	0E	ld c,N	rre (hl)	
15	?	0F	rrea	rre a	
16	(	10	djnz DIS	rl b	
17	)	11	ld de,NN	rl c	
18	>	12	ld (de),a	rl d	
19	<	13	inc de	rl e	
20	=	14	inc d	rl h	
21	+	15	dec d	rl l	
22	-	16	ld d,N	rl (hl)	
23	*	17	rla	rl a	
24	/	18	jr DIS	rr b	
25	:	19	add hl,de	rr c	
26	'	1A	ld a,(de)	rr d	
27	.	1B	dec de	rr e	
28	∅	1C	inc e	rr h	

<i>Código</i>	<i>Caracter</i>	<i>Hexa-decimal</i>	<i>Z80 assembler</i>	<i>após CB</i>	<i>após ED</i>		
29	I	1D	dec e	rr l			
30	2	1E	ld e,N	rr (hl)			
31	3	1F	rra	rr a			
32	4	20	jr nz,DIS	sla b			
33	5	21	ld hl,NN	sla c			
34	6	22	ld (NN),hl	sla d			
35	7	23	inc hl	sla e			
36	8	24	inc h	sla h			
37	9	25	dec h	sla l			
38	A	26	ld,h,N	sla (hl)			
39	B	27	daa	sla a			
40	C	28	jr z,DIS	sra b			
41	D	29	add hl,hl	sra c			
42	E	2A	ld hl,(NN)	sra d			
43	F	2B	dec hl	sra e			
44	G	2C	inc l	sra h			
45	H	2D	dec l	sra l			
46	I	2E	ld l,N	sra (hl)			
47	J	2F	cpl	sra a			
48	K	30	jr nc,DIS				
49	L	31	ld sp,NN				
50	M	32	ld (NN),a				
51	N	33	inc sp				
52	O	34	inc (hl)				
53	P	35	dec (hl)				
54	Q	36	ld (hl),N				
55	R	37	scf				
56	S	38	jr c,DIS	srl b			
57	T	39	add hl,sp	srl c			
58	U	3A	ld a,(NN)	srl d			
59	V	3B	dec sp	srl e			
60	W	3C	inc a	srl h			
61	X	3D	dec a	srl l			
62	Y	3E	ld a,N	srl (hl)			
63	Z	3F	ccf	srl a			
64	<b>RND</b>	40	ld b,b	bit 0,b	in b,(c)		
65	<b>INKEYS</b>	41	ld b,c	bit 0,c	out (c),b		
66	PI	42	ld b,d	bit 0,d	sbc hl,bc		
67	}	43	ld b,e	bit 0,e	ld (NN),bc		
68		44	ld b,h	bit 0,h	neg		
69		45	ld b,l	bit 0,l	retn		
70		46	ld b,(hl)	bit 0,(hl)	im 0		
71		47	ld b,a	bit 0,a	ld i,a		
72		48	ld c,b	bit 1,b	in c,(c)		
73		49	ld c,c	bit 1,c	out (c),c		
74		}	4A	ld c,d	bit 1,d	adc hl,bc	
75			4B	ld c,e	bit 1,e	ld bc,(NN)	
76			4C	ld c,h	bit 1,h		
77			4D	ld c,l	bit 1,l	reti	
78			4E	ld c,(hl)	bit 1,(hl)		
79			4F	ld c,a	bit 1,a	ld r,a	
80				50	ld d,b	bit 2,b	in d,(c)
81				51	ld d,c	bit 2,c	out (c),d
82			52	ld d,d	bit 2,d	sbc hl,dc	

<i>Código</i>	<i>Caracter</i>	<i>Hexa-decimal</i>	<i>Z80 assembler</i>	<i>após CB</i>	<i>após ED</i>
83	}	53	ld d,e	bit 2,e	ld (NN),de
84		54	ld d,h	bit 2,h	
85		55	ld d,l	bit 2,l	
86		56	ld d,(hl)	bit 2,(hl)	im 1
87		57	ld d,a	bit 2,a	ld a,i
88		58	ld e,b	bit 3,b	in e,(c)
89		59	ld e,c	bit 3,c	out (c),e
90		5A	ld e,d	bit 3,d	adc hl,de
91		5B	ld e,e	bit 3,e	ld de,(NN)
92		5C	ld e,h	bit 3,h	
93		5D	ld e,l	bit 3,l	
94	5E	ld e,(hl)	bit 3,(hl)	im 2	
95	5F	ld e,a	bit 3,a	ld a,r	
96	60	ld h,b	bit 4,b	in h,(c)	
97	não usado	61	ld h,c	bit 4,c	out (c),h
98		62	ld h,d	bit 4,d	sbc hl,hl
99		63	ld h,e	bit 4,e	ld (NN),hl
100		64	ld h,h	bit 4,h	
101		65	ld h,l	bit 4,l	
102		66	ld h,(hl)	bit 4,(hl)	
103		67	ld h,a	bit 4,a	rrd
104		68	ld l,b	bit 5,b	in l,(c)
105		69	ld l,c	bit 5,c	out (c),l
106		6A	ld l,d	bit 5,d	adc hl,hl
107		6B	ld l,e	bit 5,e	ld de,(NN)
108		6C	ld l,h	bit 5,h	
109		6D	ld l,l	bit 5,l	
110		6E	ld l,(hl)	bit 5,(hl)	
111		6F	ld l,a	bit 5,a	rld
112	cursor para cima	70	ld (hl),b	bit 6,b	
113	cursor para baixo	71	ld (hl),c	bit 6,c	
114	cursor para esquerda	72	ld (hl),d	bit 6,d	sbc hl,sp
115	cursor para direita	73	ld (hl),e	bit 6,e	ld (NN),sp
116	<b>GRAPHICS</b>	74	ld (hl),h	bit 6,h	
117	<b>EDIT</b>	75	ld (hl),l	bit 6,l	
118	<b>ENTER</b>	76	halt	bit 6,(hl)	
119	<b>DELETE</b>	77	ld (hl),a	bit 6,a	
120	 modo	78	ld a,b	bit 7,b	in a,(c)
121	<b>FUNCTION</b>	79	ld a,c	bit 7,c	out (c),a
122	não usado	7A	ld a,d	bit 7,d	adc hl,sp
123	não usado	7B	ld a,e	bit 7,e	ld sp,(NN)
124	não usado	7C	ld a,h	bit 7,h	
125	não usado	7D	ld a,l	bit 7,l	
126	número	7E	ld a,(hl)	bit 7,(hl)	
127	cursor	7F	ld a,a	bit 7,a	
128		80	add a,b	res 0,b	
129		81	add a,c	res 0,c	
130		82	add a,d	res 0,d	
131		83	add a,e	res 0,e	
132		84	add a,h	res 0,h	
133		85	add a,l	res 0,l	
134		86	add a,(hl)	res 0,(hl)	
135		87	add a,a	res 0,a	
136		88	adc a,b	res 1,b	

<i>Código</i>	<i>Caracter</i>	<i>Hexa- decimal</i>	<i>Z80 assembler</i>	<i>após CB</i>	<i>após ED</i>
137	■	89	adc a,c	res 1,c	
138	■	8A	adc a,d	res 1,d	
139	inverso "	8B	adc a,e	res 1,e	
140	inverso	8C	adc a,h	res 1,h	
141	inverso \$	8D	adc a,l	res 1,l	
142	inverso :	8E	adc a,(hl)	res 1,(hl)	
143	inverso ?	8F	adc a,a	res 1,a	
144	inverso (	90	sub b	res 2,b	
145	inverso )	91	sub c	res 2,c	
146	inverso >	92	sub d	res 2,d	
147	inverso <	93	sub e	res 2,e	
148	inverso =	94	sub h	res 2,h	
149	inverso +	95	sub l	res 2,l	
150	inverso —	96	sub (hl)	res 2,(hl)	
151	inverso *	97	sub a	res 2,a	
152	inverso /	98	sbc a,b	res 3,b	
153	inverso ;	99	sbc a,c	res 3,c	
154	inverso ,	9A	sbc a,d	res 3,d	
155	inverso .	9B	sbc a,e	res 3,e	
156	inverso Ø	9C	sbc a,h	res 3,h	
157	inverso 1	9D	sbc a,l	res 3,l	
158	inverso 2	9E	sbc a,(hl)	res 3,(hl)	
159	inverso 3	9F	sbc a,a	res 3,a	
160	inverso 4	A0	and b	res 4,b	ldi
161	inverso 5	A1	and c	res 4,c	cpir
162	inverso 6	A2	and d	res 4,d	inir
163	inverso 7	A3	and e	res 4,e	otir
164	inverso 8	A4	and h	res 4,h	
165	inverso 9	A5	and l	res 4,l	
166	inverso A	A6	and (hl)	res 4,(hl)	
167	inverso B	A7	and a	res 4,a	
168	inverso C	A8	xor b	res 5,b	ldd
169	inverso D	A9	xor c	res 5,c	cpdr
170	inverso E	AA	xor d	res 5,d	indr
171	inverso F	AB	xor e	res 5,e	otdr
172	inverso G	AC	xor h	res 5,h	
173	inverso H	AD	xor l	res 5,l	
174	inverso I	AE	xor (hl)	res 5,(hl)	
175	inverso J	AF	xor a	res 5,a	
176	inverso K	B0	or b	res 6,b	ldir
177	inverso L	B1	or c	res 6,c	cpir
178	inverso M	B2	or d	res 6,d	inir
179	inverso N	B3	or e	res 6,e	otir
180	inverso O	B4	or h	res 6,h	
181	inverso P	B5	or l	res 6,l	
182	inverso Q	B6	or (hl)	res 6,(hl)	
183	inverso R	B7	or a	res 6,a	
184	inverso S	B8	cp b	res 7,b	laddr
185	inverso T	B9	cp c	res 7,c	cpdr
186	inverso U	BA	cp d	res 7,d	indr
187	inverso V	BB	cp e	res 7,e	otdr
188	inverso W	BC	cp h	res 7,h	
189	inverso X	BD	cp l	res 7,l	
190	inverso Y	BE	cp (hl)	res 7,(hl)	

<i>Código</i>	<i>Caracter</i>	<i>Hexa- decimal</i>	<i>Z80 assembler</i>	<i>após CB</i>	<i>após ED</i>
191	inverso Z	BF	cp a	res 7,a	
192	""	C0	ret nz	set 0,b	
193	<b>AT</b>	C1	pop bc	set 0,c	
194	<b>TAB</b>	C2	jp nz,NN	set 0,d	
195	não usado	C3	jp NN	set 0,e	
196	<b>CODE</b>	C4	call nz,NN	set 0,h	
197	<b>VAL</b>	C5	push bc	set 0,l	
198	<b>LEN</b>	C6	add a,N	set 0,(hl)	
199	<b>SIN</b>	C7	rst 0	set 0,a	
200	<b>COS</b>	C8	ret z	set 1,b	
201	<b>TAN</b>	C9	ret	set 1,c	
202	<b>ASN</b>	CA	jp z,NN	set 1,d	
203	<b>ACS</b>	CB		set 1,e	
204	<b>ATN</b>	CC	call z,NN	set 1,h	
205	<b>LN</b>	CD	cal NN	set 1,l	
206	<b>EXP</b>	CE	adc a,N	set 1,(hl)	
207	<b>INT</b>	CF	rst 8	set 1,a	
208	<b>SQR</b>	D0	ret nc	set 2,b	
209	<b>SGN</b>	D1	pop de	set 2,c	
210	<b>ABS</b>	D2	jp nc,NN	set 2,d	
211	<b>PEEK</b>	D3	out N,a	set 2,e	
212	<b>USR</b>	D4	call nc,NN	set 2,h	
213	<b>STR\$</b>	D5	push de	set 2,l	
214	<b>CHR\$</b>	D6	sub N	set 2,(hl)	
215	<b>NOT</b>	D7	rst 16	set 2,a	
216	**	D8	ret c	set 3,b	
217	<b>OR</b>	D9	exx	set 3,c	
218	<b>AND</b>	DA	jp c,NN	set 3,d	
219	<=	DB	in a,N	set 3,e	
220	>=	DC	call c,NN	set 3,h	
221	<>	DD	prefixo de instruções usando ix	set 3,l	
222	<b>THEN</b>	DE	sbc a,N	set 3,(hl)	
223	<b>TO</b>	DF	rst 24	set 3,a	
224	<b>STEP</b>	E0	ret po	set 4,b	
225	<b>LPRINT</b>	E1	pop hl	set 4,c	
226	<b>LLIST</b>	E2	jp po,NN	set 4,d	
227	<b>STOP</b>	E3	ex (sp),hl	set 4,e	
228	<b>SLOW</b>	E4	call po,NN	set 4,h	
229	<b>FAST</b>	E5	push hl	set 4,l	
230	<b>NEW</b>	E6	and N	set 4,(hl)	
231	<b>SCROLL</b>	E7	rst 32	set 4,a	
232	<b>CONT</b>	E8	ret pe	set 5,b	
233	<b>DIM</b>	E9	jp (hl)	set 5,c	
234	<b>REM</b>	EA	jp pe,NN	set 5,d	
235	<b>FOR</b>	EB	ex de,hl	set 5,e	
236	<b>GOTO</b>	EC	call pe,NN	set 5,h	
237	<b>GOSUB</b>	ED		set 5,l	
238	<b>INPUT</b>	EE	xor N	set 5,(hl)	
239	<b>LOAD</b>	EF	rst 40	set 5,a	
240	<b>LIST</b>	F0	ret p	set 6,b	
241	<b>LET</b>	F1	pop af	set 6,c	
242	<b>PAUSE</b>	F2	jp p,NN	set 6,d	
243	<b>NEXT</b>	F3	di	set 6,e	

<i>Código</i>	<i>Caracter</i>	<i>Hexa- decimal</i>	<i>Z80 assembler</i>	<i>após CB</i>	<i>após ED</i>
244	<b>POKE</b>	F4	call p,NN	set 6,h	
245	<b>PRINT</b>	F5	push af	set 6,l	
246	<b>PLOT</b>	F6	or N	set 6,(hl)	
247	<b>RUN</b>	F7	rst 48	set 6,a	
248	<b>SAVE</b>	F8	ret m	set 7,b	
249	<b>RAND</b>	F9	ld sp,hl	set 7,c	
250	<b>IF</b>	FA	jp m,NN	set 7,d	
251	<b>CLS</b>	FB	ei	set 7,e	
252	<b>UNPLOT</b>	FC	call m,NN	set 7,h	
253	<b>CLEAR</b>	FD	prefixo de instruções usando iy	set 7,l	
254	<b>RETURN</b>	FE	cp N	set 7,(hl)	
255	<b>COPY</b>	FF	rst 56	set 7,a	

## APÊNDICE B

### Códigos indicadores

Esta tabela fornece cada um dos códigos indicadores do computador, com uma descrição geral e uma lista de instruções e funções nas quais os mesmos podem ocorrer. No Apêndice C, sob cada instrução ou função, você poderá encontrar uma descrição mais detalhada do que significa a indicação de erro.

Código	Significado	Situações		
Ø	Execução bem sucedida ou salto para uma linha de maior número que qualquer outra existente. Uma indicação Ø não altera o número da linha usado por <b>CONT</b>	qualquer	3	Subscrito fora de faixa. Caso o subscrito esteja fora de faixa (ou seja, negativo ou acima de 65535), ocorrerá o erro B.
1	A variável de controle não existe (não foi estabelecida por uma instrução <b>FOR</b> ), mas existe uma outra variável com o mesmo nome	<b>NEXT</b>	4	Espaço insuficiente na memória. Observe que o número de linha na indicação (após o /) poderá estar incompleto na tela, devido à falta de memória; assim, por exemplo, 4/2Ø poderá surgir como 4/2. Veja o capítulo 22.
2	Foi utilizada uma variável indefinida. No caso de uma variável simples, isso ocorrerá se a mesma for usada antes de ter sido referenciada por uma instrução <b>LET</b> . Para variáveis subscritas, isso acontecerá se as mesmas forem usadas antes de ser dimensionadas pela instrução <b>DIM</b> . É para uma variável de controle, isso ocorrerá caso a mesma for usada antes de ter sido definida como variável de controle por uma instrução <b>FOR</b> e se não houver nenhuma variável simples com o mesmo nome.	qualquer	5	Não há mais espaço na tela. <b>CONT</b> criará espaço, limpando a tela.
			6	Sobrecarga aritmética: os cálculos produziram um número superior a $10^{14}$ .
			7	<b>RETURN</b> sem um <b>GO-SUB</b> correspondente
			8	Você tentou um comando <b>INPUT</b> não permitido
			9	Instrução <b>STOP</b> executada. <b>CONT</b> não tentará reexecutar a instrução <b>STOP</b> .
			A	Argumento inválido para certas funções.
				variáveis subscritas
				<b>LET, INPUT, DIM, PRINT, LIST, PLOT, UNPLOT, FOR, GOSUB;</b> Às vezes, durante avaliação de funções.
				<b>PRINT, LIST, PLOT, UNPLOT.</b>
				qualquer cálculo aritmético.
				<b>RETURN</b>
				<b>INPUT</b>
				<b>STOP</b>
				<b>SQR, LN, ASN, ACS</b>

B	<p>Número inteiro fora de faixa. Quando um número inteiro é requerido, o argumento de ponto flutuante é arredondado para o número inteiro mais próximo. Caso isso esteja fora de uma faixa adequada, surge erro B.</p> <p>Para acesso a <i>array</i>, veja também indicador 3.</p>	<p><b>RUN, RAN, POKE, DIM, GOTO, GOSUB, LIST, PAUSE, PLOT, UNPLOT, CHR\$, PEEK,USR.</b> acesso a <i>array</i></p>	D	<p>1. Programa interrompido por <b>BREAK</b>.</p> <p>2. A linha <b>INPUT</b> começa com <b>STOP</b></p>	<p>ao fim de qqer. instrução ou em <b>LOAD, SAVE, LPRINT, LLIST</b> ou <b>COPY</b> <b>INPUT</b></p>
C	<p>O texto do argumento (de <i>string</i>) de <b>VAL</b> não forma uma expressão numérica válida.</p>	<p><b>VAL</b>.</p>	E	<p>Não utilizado</p>	
			F	<p>O nome fornecido para o programa é uma <i>string</i> vazia.</p>	<p><b>SAVE</b></p>

## APÊNDICE C

### O NE-Z8000 para os que já entendem Basic

#### Introdução

Se você já conhece a linguagem Basic, não deveria encontrar problemas ao utilizar o NE-Z8000. Entretanto, ele exibe algumas peculiaridades:

1. As palavras não são soletradas — possuem suas próprias teclas, como se pode verificar consultando os capítulos 2 e 5. Ao longo do texto deste Manual, tais palavras são escritas em **negrito**.
2. O NE-Z8000 não possui **READ**, **DATA** e **RESTORE** em seu Basic (veja exercício 3 do capítulo 21, que trata disso), nem funções definidas pelo usuário (**FN** e **DEF**, apesar de **VAL** poder ser utilizada algumas vezes), nem linhas de multi-instrução.
3. Os recursos para se lidar com *strings* são fáceis de compreender, mas não são padrão — veja os capítulos 20 e 21.
4. O conjunto de caracteres do NE-Z8000 é exclusivo.
5. A tela de TV não é, normalmente, mapeada em memória.
6. Se você está acostumado a utilizar **PEEK** e **POKE** em outros computadores, lembre-se de que todos os endereços serão diferentes para o NE-Z8000.

#### Velocidade

O NE-Z8000 trabalha em uma velocidade denominada *fast*, mas futuramente, mediante uma pequena adaptação, poderá também operar em *slow*.

Na modalidade chamada de *slow* (lenta), a imagem de TV é gerada continuamente, enquanto a computação é executada durante os períodos de apagamento, na parte superior e inferior da imagem.

Na modalidade *fast* (rápida), a imagem da TV é desligada durante o processamento e apresentada apenas ao fim de um programa, enquanto está esperando por dados de entrada ou durante uma pausa (veja **PAUSE**).

A modalidade *fast* é cerca de 4 vezes mais rápida que o *slow* e é de grande utilidade, especialmente em programas que exigem uma grande quantidade de processamento ou extremamente longos.

#### Teclado

Os caracteres do NE-Z8000 englobam não só os símbolos simples (letras, dígitos, etc.) como também os sinais compostos (palavras-chave, nomes de funções, etc.), sempre assinalados em **negrito**, ao longo do texto) e todos são introduzidos através do teclado. Para tornar isso possível, chegou-se a atribuir até 5 significados distintos a algumas teclas, diferenciados em parte pelo uso da tecla **SHIFT** e parte pela utilização da máquina em diferentes modalidades.

A modalidade é indicada pelo cursor, uma letra em vídeo inverso que indica onde o caractere seguinte, inserido pelo teclado, será inscrito.

A modalidade **█** (para palavras-chave) ocorre automaticamente quando a máquina aguarda um comando ou linha de programa e, através de sua posição na linha, ele sabe que deve esperar um número de linha ou uma palavra-chave. Isto no início da linha, ou alguns dígitos após o início da mesma, ou, ainda, após o **THEN**. Se não estiver combinada com a tecla **SHIFT**, a tecla seguinte será interpretada como uma palavra-chave (que estão escritas acima das teclas) ou como um dígito.

A modalidade **■** (para letras) ocorre normalmente a toda hora. Quando não está acompanhada pela tecla **SHIFT**, a tecla seguinte será interpretada como o símbolo principal contido na mesma.

Nas modalidades **█** e **■**, uma tecla combinada com **SHIFT** será interpretada como o caractere assinalado em vermelho, do canto superior direito da mesma.

A modalidade **▣** (para funções) ocorre sempre após o acionamento da tecla **FUNCTION (SHIFT + ENTER)** e tem a duração de apenas um acionamento. Essa tecla será interpretada como nome de função, que aparece escrita abaixo de cada tecla.

A modalidade **▤** (para gráficos) ocorre sempre após o acionamento da tecla **GRAPHICS (SHIFT + tecla 9)** e perdura até que seja pressionada novamente. Uma tecla não

acompanhada de **SHIFT** fornecerá vídeo inverso em sua interpretação da modalidade **I**. Uma tecla combinada com **SHIFT** fará o mesmo, desde que seja um símbolo; mas se essa tecla combinada fornecer normalmente um sinal composto, na modalidade gráfica fornecerá o símbolo para gráficos, no canto inferior direito da tecla.

#### A tela

Possui 24 linhas, cada uma com 32 caracteres, e é dividida em 2 partes. A porção superior chega no máximo a 22 linhas e exibe tanto as listagens quanto a saída dos programas; a porção inferior, que ocupa 2 linhas, no máximo, é usada para introduzir comandos, linhas de programa e dados, além de apresentar os indicadores.

Entrada via teclado: aparece na metade inferior da tela, à medida que cada caractere é digitado (símbolos simples ou compostos) sempre um pouco à esquerda do cursor. Este pode ser movido para a esquerda através de  $\circ$  (**SHIFT** + 5) e para a direita por  $\circ$  (**SHIFT** + 8); e o caractere localizado depois do cursor pode ser removido através de **DELETE** (**SHIFT** +  $\emptyset$ ). É claro que a linha inteira pode ser eliminada ao se acionar **EDIT** (**SHIFT** + 1), seguida de **ENTER**.

Sempre que **ENTER** é pressionada, a linha é executada, introduzida no programa ou empregada como dado de entrada, a menos que contenha um erro de sintaxe, caso em que aparece o símbolo **S** logo antes do mesmo.

A medida que as linhas de programação são introduzidas, uma listagem vai sendo impressa na porção superior da tela. A maneira pela qual essa listagem é introduzida está explicada em detalhes no exercício 6 do capítulo 9.

A última linha a ser introduzida é denominada **linha corrente**, sendo indicada pelo símbolo **■**; isso, porém, pode ser mudado utilizando-se as teclas  $\circ$  (**SHIFT** + 6) e  $\circ$  (**SHIFT** + 7). Se **EDIT** for acionada, a linha corrente pode ser transferida para a porção inferior da tela e lá ser editada.

Quando um comando é executado ou um programa rodado, a saída de dados aparece na porção superior da tela e lá permanece até que seja digitada uma linha de programa, ou até que **ENTER** seja acionada com uma linha vazia, ou ainda até que as teclas  $\circ$  ou  $\circ$  sejam pressionadas. Na porção inferior aparece uma indicação sob a forma  $m/n$ , onde  $m$  é um código que mostra o que ocorreu (veja apêndice B), enquanto  $n$  representa o número da última linha que foi executada — ou é  $\emptyset$  para o caso de comandos. A indicação permanece até que uma nova tecla seja acionada (e a modalidade **I** apareça).

Em determinadas circunstâncias, a tecla **SPACE** atua como **BREAK**, parando o computador com indicação D. Isto é reconhecido:

1. Ao final de uma instrução, enquanto o programa está rodando;
2. Quando o computador está procurando por um programa na fita;
3. Ou quando o computador estiver usando a impressora (ou, acidentalmente, tentar usá-la quando não está conectada).

#### O Basic

Os números são armazenados com uma precisão de 9 ou  $10^8$  dígitos. O maior número que se pode obter, no NE-

Z8000, gira em torno de  $10^{38}$ , enquanto o menor deles (positivo) é de  $4 \cdot 10^{-39}$ .

Os números podem ser armazenados no NE-Z8000 em ponto flutuante, com um byte de expoente ( $e$ , onde  $1 \leq e \leq 255$ ) e quatro bytes de mantissa ( $m$ , onde  $1/2 \leq m < 1$ ), o que representa o número  $m \cdot 2^{e-128}$ .

Já que  $1/2 \leq m < 1$ , o bit mais significativo da mantissa é sempre 1. Assim sendo, podemos trocá-lo por um bit para representar o sinal —  $\emptyset$  para números positivos, 1 para negativos.

O zero ganhou uma representação especial, nas quais todos os 5 bytes são  $\emptyset$ . As variáveis numéricas possuem nomes de tamanhos arbitrário, começando com uma letra e continuando com letras e dígitos. Os espaços são ignorados.

As variáveis de controle de loops **FOR-NEXT** tem nomes de apenas uma letra. Os *arrays* numéricos também exigem nomes de apenas uma letra, que podem ser os mesmos de uma variável simples. Podem exibir, arbitrariamente, várias dimensões, de tamanho arbitrário. Os subscritos começam em 1. As *strings* são completamente flexíveis em tamanho. O nome de uma *string* consiste de uma única letra, seguida por  $\$$ .

Os *arrays* tipo *string*, também, podem apresentar arbitrariamente as mais variadas dimensões, de tamanho arbitrário. Seu nome é composto por uma só letra, seguida de  $\$$  e pode não ser o mesmo nome de uma *string* simples. Todas as *strings* de um *array* possuem o mesmo tamanho fixo, especificado como uma dimensão extra final da instrução **DIM**. Seus subscritos começam em 1.

*Slicing*: As sub-*strings* de *strings* podem ser especificadas pelo uso de operações de *slicing*. Tais operações podem ser:

1. vazias
2. expressões numéricas
3. expressão numérica opcional para expressão numérica opcional, sendo usada para expressar uma sub-*string* por:
  - a. expressão de *string* (operação de *slicing*) ou
  - b. variável de *array string* (subscrito, ..., subscrito, *slicer*) que significa o mesmo que variável *array string* (subscrito, ..., subscrito) (*slicer*)

Em (a), suponha que a expressão de *string* tenha o valor  $s\$$ . Se o *slicer* estiver vazio, o resultado  $s\$$  será considerado uma sub-*string* de si mesmo. Se o *slicer* for uma expressão numérica com valor  $m$ , o resultado será o  $m$ .º caractere de  $s\$$  (uma sub-*string* de comprimento 1).

Se o *slicer* possuir a forma (iii), suponha que a primeira expressão numérica tenha o valor  $m$  e a segunda,  $n$ . Se  $1 \leq m \leq n \leq$  ao tamanho de  $s\$$ , o resultado é uma sub-*string* de  $s\$$ , começando no  $m$ .º caractere e terminando no  $n$ .º.

Se  $\emptyset \leq n < m$ , o resultado é uma *string* vazia. Caso contrário, surge o indicador de erro 3.

A operação de *slicing* é executada antes que as funções ou operações sejam avaliadas, a não ser que os parênteses determinem o contrário. As sub-*strings* podem ser referenciadas (veja **LET**).

O argumento de uma função não exige parênteses se é uma constante ou uma variável (possivelmente subscrita ou passada por uma operação de *slicing*).

Função	Tipo de Operando (X)	Resultado	Função	Tipo de operando	Resultado
<b>ABS</b>	Número	Magnitude absoluta	<b>PI</b>	Sem operando	$\pi$ (3,14159265...)
<b>ACS</b>	Número	arco-cosseno em radianos erro A se X não estiver entre -1 e 1.	<b>RND</b>	Sem operando	Gera uma seqüência pseudo-aleatória de números entre 0 e 1.
<b>AND</b>	Operação binária, operando da direita sempre um número Operando esquerdo numérico:	$A \text{ AND } B = \begin{cases} A & \text{se } B \neq 0 \\ 0 & \text{se } B = 0 \end{cases}$	<b>SGN</b>	Número	o sinal (0, 1 ou -1) de X.
	Operando esquerdo <i>string</i> :	$A\$ \text{ AND } B = \begin{cases} A\$ & \text{se } B \neq \emptyset \\ "" & \text{se } B = \emptyset \end{cases}$	<b>SIN</b>	Número (em radianos)	seno
<b>ASN</b>	Número	arco-seno em radianos erro A se x não estiver entre -1 e 1.	<b>SQR</b>	Número	Raiz quadrada; erro B se $X < 0$
<b>ATN</b>	Número	Arco-tangente em radianos	<b>STR\$</b>	Número	A <i>string</i> de caracteres que seria mostrada se X fosse impresso.
<b>CHRS</b>	Número	o caractere cujo código é X, arredondado para o número inteiro mais próximo. Erro B se X não estiver na faixa de 0 a 255.	<b>TAN</b>	Número em radianos	Tangente
<b>CODE</b>	String	O código do 1º caractere em X (ou 0, se X for a <i>string</i> vazia).	<b>USR</b>	Número	Chama a sub-rotina em linguagem de máquina cujo primeiro endereço é X. Ao retornar, o resultado é o conteúdo do par de registradores bc.
<b>COS</b>	Número em radianos	cosseno	<b>VAI</b>	string	Calcula X como se fosse uma expressão numérica. Erro C se X contiver um erro de sintaxe, ou fornecer o valor da <i>string</i> . Outros erros são possíveis, dependendo da <i>string</i> .
<b>EXP</b>	Número	$e^x$	<b>-</b>	Número	negação
<b>INKEY\$</b>	Sem operando	Lê o teclado. O resultado é o caractere representando (em modo <b>■</b> ) a tecla pressionada, se isto ocorrer, ou então uma <i>string</i> vazia.	Os seguintes símbolos são operações binárias (envolvendo dois números):		
<b>INT</b>	Número	Parte inteira (sempre arredondada para menos)	<b>+</b>	Adição (em números), ou concatenação (em <i>strings</i> )	
<b>LEN</b>	string	Tamanho	<b>-</b>	Subtração	
<b>LN</b>	Número	Logaritmo natural (na base e). Erro A se $X \leq 0$	<b>*</b>	Multiplicação	
<b>NOT</b>	Número	0 se $X \neq 0$ , 1 se $X = 0$ . <b>NOT</b> tem prioridade 4.	<b>/</b>	Divisão	
<b>OR</b>	Operação binária Ambos os operandos números	$A \text{ OR } B = \begin{cases} 1 & \text{se } B \neq 0 \\ A & \text{se } B = 0 \end{cases}$	<b>**</b>	Elevação a uma potência. Erro B se o operando esquerdo for negativo.	
<b>PEEK</b>	Número	<b>OR</b> tem prioridade 2. O valor do byte da memória cujo endereço é X (arredondado para o inteiro mais próximo). Erro B se X não estiver na faixa de 0 a 65535.	<b>=</b>	igual	
			<b>&gt;</b>	maior que	
			<b>&lt;</b>	menor que	
			<b>&lt;=</b>	menor ou igual a	
			<b>&gt;=</b>	maior ou igual a	
			<b>&lt;&gt;</b>	diferente de	
			} Ambos os operandos devem ser do mesmo tipo. O resultado é um número: 1 se a comparação for correta e 0 se não for.		
			Funções e operações têm as seguintes prioridades:		
			<b>Operações</b>	<b>Prioridade</b>	
			Subscrição e <i>slicing</i>	12	
			todas funções exceto <b>NOT</b> e menos unário	11	
			<b>**</b>	10	
			Menos unário	9	
			<b>*, /</b>	8	
			<b>+, - (- BINÁRIO)</b>	6	
			<b>=, &gt;, &lt;, &lt;=, &gt;=</b>	5	

NOT	4
AND	3
OR	2

### INSTRUÇÕES

nesta tabela.

$\alpha$	representa apenas uma letra
V	representa uma variável
x, y, z	representam expressões numéricas
m, n	representam expressões numéricas que são arredondadas para o inteiro mais próximo.
e	representa uma expressão
f	representa uma expressão e/ valor de <i>string</i>
s	representa uma instrução

Note que expressões arbitrárias são permitidas em qualquer lugar (exceto para número de linha no início da instrução).

Todas as instruções, exceto **INPUT**, podem ser usadas tanto como comando como no programa.

**CLEAR** Cancela todas as variáveis, liberando o espaço ocupado por elas.

**CONT** Supondo que a/b foi a última denotação com não-zero, então **CONT** tem o efeito  
**GOTO b** se  $a \neq 9$   
**GOTO b + 1** se  $a = 9$  (instrução **STOP**)

**CLS** Limpa o arquivo da tela. Veja capítulo 27.

**COPY** Envia uma cópia da tela para a impressora, se estiver acoplada; caso contrário, não faz nada. Denotação D, se **BREAK** for pressionado.

**DIM**  $\alpha$  (n1, ..., nk) Cancela qualquer *array* com o nome  $\alpha$  e define um *array* de números com k dimensões n1, ..., nk. Zera todos os elementos. Erro 4 ocorre se não há espaço na memória para o *array*. Um *array* é indefinido, até ser dimensionado em uma instrução **DIM**.

**DIM**  $\alpha$  \$ (n1, ..., nk) Cancela qualquer *array* ou *string* com nome  $\alpha$  \$ e define um *array* de caracteres com k dimensões n1, ..., nk "Branqueia" todos os elementos. Pode ser considerada uma *array* de *strings* de tamanho fixo nk, com k-1 dimensões n1, ..., nk-1.

Erro 4 ocorre se não há espaço na memória para o *array*. Um *array* é indefinido até que seja dimensionado por uma instrução **DIM**.

**FOR**  $\alpha = x$  TO y **FOR**  $\alpha = x$  TO y STEP 1

**FOR**  $\alpha = x$  TO y STEP z Cancela qualquer variável  $\alpha$  e define uma variável de controle com valor x, limite y, passo z, e o endereço de LOOP 1 mais o número da linha da instrução **FOR** (-1 se for um comando). Verifica se o valor inicial é maior (se passo  $\geq 0$ ) ou menor (se passo  $< 0$ ) que o limite, e em caso afirmativo, volta para a instrução **NEXT**  $\alpha$ , no início da linha. Veja **NEXT**  $\alpha$ . Erro 4 ocorre se não houver espaço na memória para a variável de controle.

**GOSUB** n Coloca o número da linha da instrução **GOSUB**

em stack. A seguir, age como **GOTO** n. Erro 4 pode ocorrer se não houver **RETURN**s suficientes.

**GOTO** n Salta para linha n (ou, se não houver, para primeira após a mesma).

**IF x THEN s** Se x for verdadeiro ( $\neq$  zero), então S é executado. O formato "**IF x THEN** número da linha" não é permitido.

**INPUT** V Para e espera que seja feita uma entrada de dados. O arquivo de imagem é impresso no modo **FAST**. **INPUT** não pode ser usado como comando, pois ocorrerá erro 8.

Se o primeiro caractere em uma linha de **INPUT** é **STOP**, o programa pára com denotação D.

**LET** v = c Atribui o valor e à variável v. **LET** não pode ser omitido. Uma variável simples é indefinida, até figurar em uma instrução **LET** ou **INPUT**. Se v é uma variável string subscrita, então a atribuição é procrustiana: o valor e da *string* é truncado ou preenchido com zeros à direita, para ter o mesmo tamanho que a variável v.

**LIST** **LIST**  $\emptyset$

**LIST** n Lista o programa na televisão, iniciando na linha n, e faz de n a linha corrente. Erros 4 ou 5, se a listagem for muito longa para a tela. **CONT** fará exatamente o mesmo, novamente.

**LLIST** **LLIST**  $\emptyset$

**LLIST** n Como **LIST**, porém usando a impressora e não a TV. Não faz nada se não houver impressora; pára com denotação D, se **BREAK** for pressionado. Procura o programa chamado f na fita e carrega-o com suas variáveis. Se f = "", então carrega o primeiro programa encontrado.

Se **BREAK** for pressionado, então

(I) se nenhum programa for lido na fita, pára com denotação D e o programa antigo;

(II) se uma parte do programa foi lida, então executa **NEW**.

**LPRINT..** Como **PRINT**, mas usando a impressora. Uma linha do texto é enviada à impressora:

(I) Quando a impressão salta de uma linha para a próxima;

(II) Após um **LPRINT** que não termina em uma vírgula ou um ponto-e-vírgula.

(III) Quando uma vírgula ou item **TAB** requer uma nova linha.

(IV) No fim do programa, se há algo que não foi impresso. Num item **AT**, apenas o número da coluna tem efeito; o número da linha é ignorado. Um item **AT** nunca manda uma linha de texto para a impressora.

Não há efeito algum se a impressora estiver ausente.

Pára com denotação D, se for pressionado **BREAK**.

**NEW** Inicia o sistema BASIC, cancelando o programa e as variáveis e usando a memória até (mas não

incluindo) o byte cujo endereço está na variável RAMTOP (bytes 16388 e 16389).

**NEXT**  $\alpha$  (I) Encontra a variável de controle  $\alpha$ .  
(II) Adiciona o passo ao seu valor  
(III) Se o passo  $\geq \theta$  e o valor  $>$  limite; ou se o passo  $< \theta$  e o valor  $<$  limite, então salte para a linha de loop.

Erro 1 se não há variável de controle  $\alpha$ .

**PAUSE**  $n$  Pára de computar e exhibe o arquivo de imagem de  $n$  quadros (a  $5\theta$  quadros por segundo) ou até uma tecla ser pressionada.

$\theta \leq n \leq 65535$ , ou então erro B.

Se  $n \geq 32767$ , então a pausa não é temporizada, mas dura até uma tecla ser pressionada.

**PLOT**  $m, n$  Faz com que o elemento ( $|m|$ ,  $|n|$ ) fique preto; move a posição de impressão para após o elemento.  $\theta \leq |m| < 63, \theta \leq |n| \leq 43$ , ou erro B.

**POKE**  $m, n$  Escreve o valor  $n$  do byte de endereço  $m$ .  $\theta \leq m \leq 65535$ ,  $-255 \leq n \leq 255$ , ou então erro B.

**PRINT...** O "... " é uma seqüência de itens **PRINT**, separados por vírgula ou ponto-e-vírgula. Eles estão escritos no arquivo de imagem na televisão.

A posição (linha e coluna) onde o próximo caractere deve ser impresso é chamado de posição de impressão (**PRINT**)

Um item **PRINT** pode ser:

(I) Vazio, isto é, nada;

(II) Uma expressão numérica

Primeiro, um sinal menos é impresso se o valor é negativo. Em seguida, atribui a  $X$  o módulo do valor.

Se  $X \leq 1\theta^{-5}$  ou  $X \geq 1\theta^{13}$ , então a impressão usa notação científica. A mantissa tem até 8 dígitos e o ponto decimal (ausente em caso de 1 dígito) está após o primeiro. O expoente é  $E$ , seguido de  $+$  ou  $-$ , seguido de mais 2 dígitos. Em caso contrário,  $X$  é impresso em notação decimal ordinária de até oito dígitos significativos e sem zeros após o ponto decimal. Um ponto decimal no início é sempre seguido de um zero.

$\theta$  é impresso com um único dígito  $\theta$ .

(III) Uma expressão *string*

Os símbolos nas *strings* são expandidas, provavelmente com um espaço antes ou depois. O caractere de aspas é impresso como  $''$ . Caracteres não usados e caracteres de controle são impressos como  $?$ .

(IV) **AT**  $m, n$

A posição de impressão é mudada para linha  $m$  (contando do topo), coluna  $n$  (contando da esquerda).  $\theta \leq |m| \leq 21, \theta \leq |n| \leq 31$ , ou então erro B. Se  $|m| = 22$  ou  $23$ , erro 5.

(V) **TAB**  $n$

$n$  é reduzido para módulo 32. Então, a posição de impressão é movida para coluna  $n$ , permanecendo na mesma linha, a menos que isso envolva retorno na mesma linha; neste caso, move-se para próxima linha.

$\theta \leq n \leq 255$ , ou então erro B.

Um ponto-e-vírgula entre 2 itens imobiliza a posição de impressão de forma que o 2º item siga logo após o primeiro.

Uma vírgula, por outro lado, move a posição de impressão de no mínimo um lugar; e, após isso, tantos quanto forem necessários para deixá-la na coluna  $\theta$  ou 16, introduzindo uma nova linha, se necessário. Ao fim de uma instrução **PRINT**, se não terminar em um ponto-e-vírgula ou vírgula, uma nova linha será colocada.

Erro 4 (fora da memória) pode ocorrer com 3k ou menos memória.

Erro 5 significa que a tela está repleta.

Em ambos os casos, a solução é **CONT**, que limpará a tela e continuará.

**RAND** **RAND**  $\theta$

**RAND**  $n$  Atribui valor à variável do sistema (chamado SEED), usada para gerar o próximo valor de **RND**. Se  $n \neq \theta$ , o SEED recebe o valor  $n$ ; se  $n = \theta$  lhe é dado um valor de uma outra variável do sistema (chamada FRAMES) que conta os quadros impressos até então na televisão, o que deve ser bastante randômico.

Erro B ocorre se  $n$  não estiver na faixa de  $\theta$  a 65535.

**REM...** Sem efeito, "... " pode ser qualquer seqüência de caracteres, exceto **ENTER**

**RETURN** Retira o número da linha do stack de **GOSUB** e salta para a linha seguinte. Erro 7 ocorre quando não há número de linha no stack.

Há erros no programa; **GOSUB** não está propriamente balanceado pelos **RETURNS**.

**RUN** **RUN**  $\theta$

**RUN**  $n$  **CLEAR**, e então **GOTO**  $n$

**SAVE**  $f$  Grava o programa e variáveis na fita e chama-os de  $f$ .

**SAVE** não deve ser usado em uma rotina **GOSUB**.

Erro F ocorre se  $f$  for uma *string* vazia, o que não é permitido.

**SCROLL** Gira o arquivo de imagem uma linha para cima perdendo a linha do topo e criando outra embaixo.

**STOP** Finaliza o programa com denotação 9. **CONT** reiniciará na próxima linha.

**UNPLOT**  $m, n$  Como **PLOT**, porém "branqueia" o elemento de impressão.

# Manual do NE-Z8000 — Erratas

## Capítulo II

Página 6, item 5, onde se lê  
"No nosso caso, a tecla..."

*leia-se*

"No nosso caso, a tela..."

## Capítulo IV

Página 10, 2ª coluna, 1º exemplo, onde se lê—  
"PRINT 2.34 — E2"

*leia-se*

"PRINT 2.34E—2"

## Capítulo V

Página 11, 1º parágrafo, onde se lê  
"... você deve pressionar a tecla FUNCTION  
SHIFT ENTER)."

*leia-se*

"... você deve pressionar a tecla FUNCTION, que é  
obtida pressionando-se SHIFT e ENTER  
ao mesmo tempo."

## Capítulo VIII

Página 18, 1ª coluna, 3º parágrafo, eliminar  
o símbolo  $\boxtimes$

## Capítulo IX

Página 21, 1ª coluna, 4º parágrafo, onde se lê  
"... imprimindo-as, como seus tamanhos."

*leia-se*

"... imprimindo-as com seus tamanhos."

## Capítulo XI

Página 27, 1ª coluna, linha 40 do programa, onde se lê  
40 IF <256 THEN GO TO 20

*leia-se*

40 IF A<256 THEN GO TO 20

## Capítulo XII

Página 30, 1ª coluna, 2ª programa, onde se lê  
60 LET COUNT = CONT + 1

*leia-se*

60 LET COUNT = COUNT + 1

Página 31, 1ª coluna, onde se lê

20 FOR C = 1 TO STEP 3/2

*leia-se*

20 FOR C = 1 TO 5 STEP 3/2

Página 31, 1ª coluna, 1º parágrafo, onde se lê

"C assumirá os valores 1, 2, 5 e 4."

*leia-se*

"C assumirá os valores 1, 2.5 e 4."

## Capítulo XIII

Página 32, 2ª coluna, 2º parágrafo, onde se lê  
"Você terá 3 variáveis: J, P e POL..."

*leia-se*

Você terá 3 variáveis: J, P e POL..."

## Capítulo XIX

Página 46, 2ª coluna, 1º parágrafo, onde se lê  
"... até o canto inferior esquerdo da tecla..."

*leia-se*

"... até o canto inferior esquerdo da tela..."

## Capítulo XX

Página 47, 1ª coluna, 4º parágrafo, onde se lê  
"ABCDEF" (5 TO >)

*leia-se*

"ABCDEF" (5 TO 7)

2ª coluna, 1ª linha, onde se lê

"Dá erro 3 (erro de subscripto) porque, desde que a *string*  
contenha apenas 6 caracteres > é maior..."

*leia-se*

"Dá erro 3 (erro se subscripto) porque, como a *string* contém  
apenas 6 caracteres, 7 é demais..."

2ª coluna, 1º exemplo, onde se lê

"ABCDEF" (8 TO >)

*leia-se*

"ABCDEF" (8 TO 7)

2ª coluna, 3º parágrafo, onde se lê

LET \$ = "ESTÁ MUITO FORA"

*leia-se*

LET A\$ = "ESTÁ MUITO FORA"

## Capítulo XXII

Página 51, 2ª coluna, 2º parágrafo, onde se lê  
"Tudo desaparecerá, mas o programa ainda se encontra no  
computador, como você poderá comprovar, cancelando a  
linha 10, usando o e o . Agora digite:"

*leia-se*

"Você verá que o último caractere X irá desaparecer. Agora  
digite ENTER e tudo desaparecerá; o programa, porém, ainda  
se encontra no computador, como você pode comprovar  
cancelando a linha 10 e usando as teclas e para verificar  
as linhas do mesmo."

## Capítulo XXIII

Página 54, 2ª coluna, 1º parágrafo, onde se lê  
"Um byte tem sempre 18 bits..."

*leia-se*

"Um byte tem sempre 8 bits..."

## Capítulo XXV

Página 57, 1ª coluna, 5º parágrafo, onde se lê  
"... os registradores a', f', x, iy;"

*leia-se*

"... os registradores a', f', ix, iy;"

## Capítulo XXVI

Página 59, 2ª coluna, 2º parágrafo, onde se lê  
"... usando o caracteres CHS 126..."

leia-se

"... usando o caractere **CHRS** 126..."

2ª coluna, 4º parágrafo, onde se lê

"... consiste de apenas 24 **ENTERS**."

leia-se

"... consiste de apenas 25 **ENTERS**."

### Capítulo XXVII

Página 62, onde se lê

N1 16390 MODE Especifica cursor **[K]**, **[L]**, **[F]** ou **[G]**  
Linha da instrução que está sendo...

leia-se

N1 16390 MODE Especifica cursor **[K]**, **[L]**, **[F]** ou **[G]**

N2 16391 PPC Número da linha da instrução que  
está sendo...

Mesma página, na primeira variável S1, substituir ROM  
por EPROM.

Mesma página, na 3ª variável S2, leia-se "Endereço do  
caractere precedendo o indicador **[S]**."

Mesma página, substituir **STKKEND** e **BREG** por  
**STKEND** e **BERG**.

Mesma página, na variável SN2 de nº 16415, substituir  
**MEMOT** por **MEMBOT**.

### Apêndice C

Página 73, 2ª coluna, 8º parágrafo, onde se lê

"Se  $1 \leq m \leq n \leq$  o tamanho de **s**..."

leia-se

"Se  $1 \leq m \leq n \leq$  ao tamanho de **s**..."

Página 74, instrução **NOT**, onde se lê

$\theta$  se  $X \neq \theta$ . **NOT** tem..."

leia-se

" $\theta$  se  $X \neq \theta$ , 1 se  $X = 0$ . **NOT** tem..."

Página 75, 1ª coluna, onde se lê

**DEM**  $\alpha$  **S** (n1, ... nk) Cancela qualquer *array* com  
o nome  $\alpha$  ..."

leia-se

**DIM**  $\alpha$  (n1, ... nk) Cancela qualquer *array* com o nome  $\alpha$  ..."

USE O CUPOM ABAIXO PARA FAZER O SEU PEDIDO:

Se o seu pedido do NE-Z8000 chegar à Filcores ou a qualquer dos seus distribuidores autorizados dentro de 10 dias você ganha de presente uma fita com programas para o seu NE-Z8000. Aproveite! Peça ainda hoje!

- Sim, quero receber os seguintes itens:
- Computador Pessoal NE-Z8000 ..... Cr\$ 69.900,00
  - Expansão de Memória 16Kb ..... Cr\$ 29.900,00
  - Fita com 14 Programas (1Kb) ..... Cr\$ 1.000,00
  - Fita com 5 Programas (16Kb) ..... Cr\$ 1.000,00
  - Manual NE-Z8000 (extra) ..... Cr\$ 2.000,00
- Total ..... Cr\$

Forma de Pagamento:  Cheque nominal  
 Reembolso Varig  
 Vale Postal

Nome ..... n.º .....

Endereço ..... CEP .....

Bairro ..... Estado .....

Cidade ..... Data Nascimento ..... / ..... / .....

CPF (CGC) ..... Tel. ....

Data do Pedido ..... / ..... / .....

Assinatura .....



Vendas e Assistência Técnica  
 Filcores Importação e Representações Ltda.  
 Rua Aurora, 165  
 CEP 01209 - São Paulo - SP  
 Tel.: (011) 223-7388

Distribuidores Autorizados Filcores,

- SÃO PAULO — FILCORES IMP. REPRESENTAÇÃO LTDA - Tel. 223-7388 — SO KIT - Tel. 221-4747 • A.B.C. — RÁDIO ELÉTRICA SANTISTA LTDA - Tel. 449-6986 — SANTO ANDRÉ — SÃO CAETANO DO SUL - Tel. 443-3299 — S. B. CAMPO — INCORCOMPONENTES ELÉTRONICOS LTDA - Tel. 449-1357 e 449-2411 — SANTO ANDRÉ - Tel. 446-3877 — UTINGA - Tel. 271-7028 • BELO HORIZONTE — ELETRICARAO IRMÃOS MALACCO LTDA - Tel. 201-2921 — KEMITRON LTDA - Tel. 226-8524 — ELETRO LTDA - Tel. 201-6552 • BLUMENAU — COPEL COM. DE PEÇAS ELÉTRICAS LTDA - Tel. 22-8602 • BRASÍLIA — SIMÃO ENG. ELETRÔNICA LTDA - Tel. 246-1516 — ELETRÔNICA YAPA LTDA - Tel. 22-4-0556 • CAMPINAS — BRASITONE - Tel. 31-1756 e 31-9195 • CAMPO GRANDE — ELETRÔNICA CONCORD LTDA - Tel. 383-4451 e 383-5752 • CAXIAS DO SUL — ELETRÔNICA CENTRAL - Tel. (054) 221-2389 e 221-4989 • CURITIBA — SEFAR LTDA - Tel. 233-6731 — ELETRÔNICA MODELO LTDA - Tel. 233-5033 • COMERCIAL RADIO TV UNIVERSAL LTDA - Tel. 223-4944 • RG SUL — ELETRÔNICA MUTTONI • FLORIANÓPOLIS — ELETRÔNICA RAUJAR LTDA - Tel. 463271 • FORTALEZA — ELETRÔNICA APOLO - Tel. 226-0770 • GOIÂNIA — KITELECOM E RE. PRES. DE KITS E COM. ELETRON. LTDA • JOÃO PESSOA — ELETRON. PEÇAS - Tel. 221-5098 • LONDRINA — KATSUMI HAYAMA & CIA. LTDA - Tel. 23-6770 • MACEIO — ELETRÔNICA ALAGANA LTDA - Tel. 223-4238 • MANAUS — COMERCIAL REZERRA - Tel. 252-5363 • MOGI DAS CRUZES — COMPEL COMPONENTES ELÉTRONICOS - Tel. 469-6554 e 469-6507 • NATAL — SOMA-TEL SOC. DE MATERIAS ELÉTRONICOS LTDA - Tel. 223-2153 • PIRACICABA — ELETRÔNICA PAUMAR LTDA - Tel. 22-7325 • PORTO ALEGRE — DIGITAL COMPONENTES ELÉTRONICOS LTDA - Tel. 24-1411 — IMAN IMPORTADORA - Tel. 24-0948 • RECIFE — BARTO REPIES. COM. LTDA - Tel. 224-3699 • RIBEIRÃO PRETO — A RADIO LAR - Tel. 25-4206 • RIO DE JANEIRO — DELTRONIC.COM DE EQUIP. ELÉTRONICOS LTDA - Tel. 252-2640 e 252-5334 — REI DAS VALVULAS ELÉTRONICAS LTDA - Tel. 221-7590 • SALVADOR — ELETRÔNICA SALVADOR COMÉRCIO E IMPORTAÇÃO LTDA - Tel. 243-7226 e 243-8940 — TV PEÇAS LTDA - Tel. 242-2033 • ELETRÔNICA SÃO JORGE - Tel. 226-3908 • SÃO VICENTE — ELETRÔNICA ELECTRODIT - Tel. 68-8046 • SÃO JOSÉ DOS CAMPOS — REI DOS TRANSISTORES - Tel. 21-2869 • SOROCABA — ELETRÔNICA APOLLO LTDA - Tel. 32-8046 • VITÓRIA — CASA STRAUCH - Tel. 223-4057 — ELETRÔNICA YUNG LTDA - Tel. 223-1345



# COLOQUE UM COMPUTADOR EM SUA CASA

# NE-Z8000

O Computador Pessoal.

# O coração de um sistema que cresce com você.

Se você estava pensando em comprar um computador, este é o momento para fazê-lo.

O novo NE-Z8000 é o computador pessoal mais potente e fácil de usar, na sua categoria, jamais oferecido, e por um preço igual ao de um televisor ou aparelho de som.

De fato, o NE-Z8000 com 8k de BASIC. Estendido oferece recursos somente encontrados em computadores 4 ou 5 vezes mais caros.

Veja o que você terá disponível:

- String multidimensional e matrizes numéricas.
- Funções matemáticas e científicas com precisão de 8 casas decimais.
- Teclado de um único toque para entrada de palavras chaves tais como PRINT, RUN, LIST, etc.
- Detecção automática de erro de sintaxe e fácil correção.
- Função Randômica, útil para jogos e aplicativos sérios.
- POKE, PEEK eUSR para inserir, fornecer conteúdo e ter acesso a sub-rotinas em linguagem de máquina.
- Facilidades gráficas para programas estatísticos e jogos animados.

COMANDOS  
INSTRUÇÕES

ARITMÉTICA  
FUNÇÕES

FUNÇÕES STRING

NÚMEROS

VARIÁVEIS NUMÉRICAS

VARIÁVEIS STRING

RECURSOS ESPECIAIS

LIST, LOAD, NEW, RUN, SAVE  
PRINT, INPUT, LET, GOTO, GOSUB/  
RETURN, FOR/NEXT, IF/THEN,  
STEP, TAB.

ABS, RND, PI.

INT, COS, SIN, TAN, ARCCOS,  
ARCSIN, ARCTAN, EXP, LOG, SGN,  
SQR.

CHRS, LEN, ASC (CODE), STRS, VAL,  
INKEY.

Ponto flutuante até  $\pm 10^{\pm 38}$ .

An — Zn, n = qualquer string alfa-  
numérico (conjunto de letras e números).  
AS à Z\$.  
POKE, PEEK eUSR.

## EXPANSÃO DE MEMÓRIA NEX 16K

Desenhada para harmonizar com o seu NE-Z8000, ela é encaixada no conector existente na parte traseira do computador, multiplicando sua armazenagem de dados/programa por 16!

Com o NEX 16k você poderá rodar programas mais longos e complexos, como o sofisticado *CONTA CORRENTE*.

## NOVO SOFTWARE

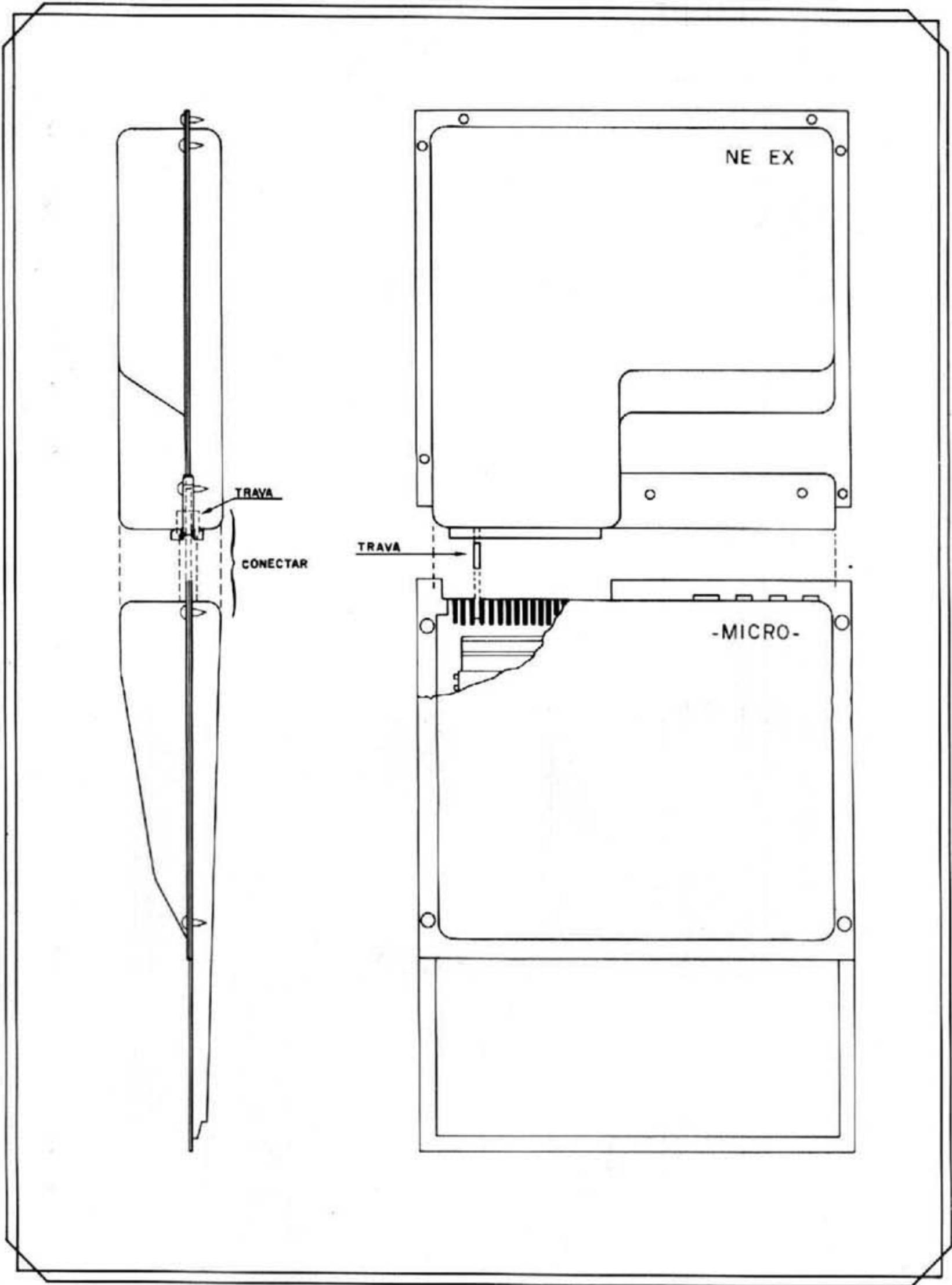
A Filres põe à sua disposição também sua biblioteca de programas para o NE-Z8000 oferecendo programas educacionais, jogos animados, cálculos científicos, contabilidade doméstica, etc., gravados em fitas especiais para você usar com o seu NE-Z8000. Ao pedir suas fitas não esqueça de especificar a memória de seu computador: 1Kb ou 16Kb. As fitas para 1Kb vêm com 14 programas variados e as fitas para 16Kb com 5 programas.

## CARACTERÍSTICAS TÉCNICAS DO NE-Z8000

- Possui interpretador de linguagem Basic residente em ROM de 8K Bytes.
- Microprocessador Z80A a 3,6 MHz.
- Memória RAM de 1K Byte, expansível para 16K Bytes.
- Teclado com 40 teclas contendo 154 funções.
- Fonte de alimentação de 9 VCC e 600 mA (acompanha o micro).

O NE-Z8000 é muito cômodo para usar. Ele se acopla à antena de qualquer aparelho de televisão. E você pode usar o seu gravador cassete para arquivar ou recuperar programas pelo nome.

O manual do NE-Z8000 vem com 76 páginas de guia de programação e de operação, elaborado para iniciantes e experimentados em computadores.



## INSTRUÇÕES DE UTILIZAÇÃO DO MÓDULO NEX-16K

Este conjunto vem acompanhado de uma outra fonte de alimentação, que deverá ser usada sempre que a expansão estiver conectada ao computador (no lugar da fonte que acompanha o micro).

Este módulo expande a capacidade de memória de seu NE-Z8000 (ou NE-Z80) para 16 kbytes (sendo 1 k, realmente, igual a 1024 unidades de armazenamento de dados). Para colocá-lo em operação, basta encaixá-lo no conector traseiro de seu computador, previsto para esse fim, conforme mostra a ilustração em anexo.

### Seqüência para a conexão do NEX-16K ao computador:

1. Com o micro desligado, encaixe os contatos aparentes de sua parte traseira no conector correspondente da expansão (é conveniente realizar essa operação sobre uma superfície plana, tal como uma mesa);

2. Observe que existe uma pequena trava no conector, correspondendo a uma pequena ranhura na placa do computador. Para conexão e encaixe perfeitos, é necessário que a ranhura e a trava coincidam com precisão.

Apesar dos contatos de ambas as unidades (micro e expansão) serem de alta qualidade e banhados a ouro, em alguns casos poderá haver mau contato entre eles. Se isto ocorrer, primeiramente desacople a expansão NEX-16K e, depois, com uma borracha macia e branca, limpe os contatos do computador. Procure manter o módulo de expansão sempre ligado ao micro, para não haver fadiga do conector, devido a constantes esforços mecânicos.

Caso ainda persistam algumas dúvidas quanto à capacidade de memória, sugerimos a execução de um pequeno programa em linguagem de máquina, que testa as memórias RAM e, ao mesmo tempo, indica o espaço das mesmas que está disponível. Eis o programa:

```
1 POKE 18000, 33
2 POKE 18001, 11
3 POKE 18002, 0
4 POKE 18003, 57
5 POKE 18004, 68
6 POKE 18005, 77
7 POKE 18006, 201
8 PRINT (USR (18000) — 16373)/ 1024 ; "K"
```

Ao rodar o programa sem o módulo, a resposta deverá ser "1 K" e, com ele, "16 K".

**Obs.: Esse programa mostra apenas o espaço de memória existente, e não o espaço livre para programas.**